

LNCS 4443

Ramamohanarao Kotagiri
P. Radha Krishna
Mukesh Mohania
Ekawit Nantajeewarawat (Eds.)

Advances in Databases: Concepts, Systems and Applications

12th International Conference on Database Systems
for Advanced Applications, DASFAA 2007
Bangkok, Thailand, April 2007, Proceedings

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Ramamohanarao Kotagiri
P. Radha Krishna Mukesh Mohania
Ekawit Nantajeewarawat (Eds.)

Advances in Databases: Concepts, Systems and Applications

12th International Conference on Database Systems
for Advanced Applications, DASFAA 2007
Bangkok, Thailand, April 9-12, 2007
Proceedings

Volume Editors

Ramamohanarao Kotagiri
The University of Melbourne
Department of Computer Science and Software Engineering
Victoria 3010, Australia
E-mail: kotagiri@unimelb.edu.au

P. Radha Krishna
Institute for Development and Research in Banking Technology
Masab Tank, Hyderabad 500 057, Andhra Pradesh, India
E-mail: prkrishna@idrbt.ac.in

Mukesh Mohania
IBM India Research Laboratory
Institutional Area, Vasant Kunj, New Delhi 110 070, India
E-mail: mkmukesh@in.ibm.com

Ekawit Nantajeewarawat
Thammasat University - Rangsit Campus
Sirindhorn International Institute of Technology
Pathum Thani 12121, Thailand
E-mail: ekawit@siit.tu.ac.th

Library of Congress Control Number: 2007923774

CR Subject Classification (1998): H.2, H.3, H.4, H.5, J.1

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-71702-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-71702-7 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2007
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12043323 06/3142 5 4 3 2 1 0

Preface

The 12th International Conference on Database Systems for Advanced Applications (DASFAA), organized jointly by the Asian Institute of Technology, National Electronics and Computer Technology Center and Sirindhorn International Institute of Technology, sought to provide information to users and practitioners of database and database systems on advanced applications.

The DASFAA conference series has already established itself and it continues to attract, each year, participants from all over the world. In this context, it may be recalled that the previous DASFAA conferences were successfully held in Seoul, Korea (1989), Tokyo, Japan (1991), Daejeon, Korea (1993), Singapore (1995), Melbourne, Australia (1997), Taiwan, ROC (1999), Hong Kong (2001), Kyoto, Japan (2003), Jeju Island, Korea (2004), Beijing, China (2005) and Singapore (2006). Thailand had the opportunity to host this prestigious and important international conference and join the league.

This conference provides an international forum for academic exchanges and technical discussions among researchers, developers and users of databases from academia, business and industry. DASFAA focuses on research in database theory, development of advanced DBMS technologies and their advanced applications. It also promotes research and development activities in the field of databases among participants and their institutions from Pacific Asia and the rest of the world .

This proceedings volume puts together 112 accepted papers from more than 18 countries in the areas of XML Databases, Mobile Databases, Query Language, Query Optimization and Data Mining etc., of which 68 are full papers, 24 are short papers, 17 are posters and 3 are industrial track papers. The conference received 375 submissions and such a rigorous selections helped retain DASFAA's reputation as a highly selective conference that publishes only quality research.

We are delighted to feature two invited talks from Guy M. Lohman, IBM Almaden Research Center, and Masaru Kitsuregawa, University of Tokyo. DASFAA 2007 also featured an excellent tutorial program covering three tutorials related to Matching Words and Pictures, Time Series Databases, XML Databases and Streams. In addition, there were three demonstrations, panel sessions and two workshops.

The members of the DASFAA Organizing Committee worked extremely hard to make this conference a success. The members of the Program Committee, consisting of renowned data management experts, undertook the arduous task of reviewing all the submitted papers and invested their valuable time and expertise, despite their extremely tight schedules. We would like to thank all the reviewers who very carefully reviewed the papers on time, the authors who submitted their papers and all the participants.

We are grateful to Alfred Hofmann and the staff of Springer for their support in publishing these proceedings.

The conference was sponsored by IBM, Thailand, the Database Society of Japan, Korea Information Science Society, National Electronics and Computer Technology Center and Software Industry Promotion Agency.

April 2007

Ramamohanarao Kotagiri
P. Radha Krishna
Mukesh Mohania
Ekawit Nantajeewarawat

DASFAA 2007 Conference Organization

Conference Chair

Vilas Wuwongse

Asian Institute of Technology, Thailand

Program Committee Co-chairs

Ramamohanarao Kotagiri
Mukesh Mohania
Ekawit Nantajeewarawat

University of Melbourne, Australia
IBM India Research, India
Sirindhorn International Institute of Technology,
Thammasat University, Thailand

Demo Co-chairs

Mizuho Iwaihara
Xuemin Lin

Kyoto University, Japan
University of New South Wales, Australia

Industrial Co-chairs

Prasan Roy
Masashi Tsuchida

IBM India Research, India
Software Division, Hitachi, Ltd., Japan

Panel Co-chairs

Sourav Bhownick
Masaru Kitsuregawa

NTU, Singapore
University of Tokyo, Japan

Tutorial Committee Co-chairs

Tharam Dillon
Haruo Yokota

University of Technology, Sydney, Australia
Tokyo Institute of Technology, Japan

Publication Chair

P. Radha Krishna

Institute for Development and Research in
Banking Technology, India

Publicity Co-chairs

Chin-Wan Chung
Qing Li

KAIST, Korea
City University of Hong Kong, PRC

Regional Chairs

Asia	Yasushi Kiyoki	Keio University, Japan
Australia and New Zealand	Millist Vincent	University of South Australia, Australia
Europe	Michael Schrefl	University of Linz, Austria
USA	Sanjay Madria	University of Missouri-Rolla, USA

Local Arrangements Chair

Suranart Tanvejsilp	NECTEC, Thailand
---------------------	------------------

Program Committee

Akiyo Nadamoto	NICT, Japan
Amol Deshpande	University of Maryland at College Park, USA
Anirban Mondal	University of Tokyo, Japan
Arkady Zaslavsky	Monash University, Australia
Arnd Christian Konig	Microsoft Research, USA
Atsuyuki Morishima	University of Tsukuba, Japan
Bala Iyer	IBM, USA
Balaraman Ravindran	IIT Madras, India
Barbara Catania	University of Genoa, Italy
Charnyot Pluempiwiriyawej	Mahidol University, Thailand
Chiang Lee	National Cheng Kung University, Taiwan
Cholwich Nattee	Thammasat University, Thailand
Chutiporn Anutariya	Shinawatra University, Thailand
Dan Lin	National University of Singapore, Singapore
David Embley	Brigham Young University, USA
David Taniar	Monash University, Australia
Dimitrios Gunopulos	UCR, USA
Egemen Tanin	University of Melbourne, Australia
Elena Ferrari	University of Insubria, Italy
Ernesto Damiani	University of Milan, Italy
Evaggelia Pitoura	University of Ioannina, Greece
Gao Cong	University of Edinburgh, UK
Gill Dobbie	University of Auckland, New Zealand
Gunther Pernul	University of Regensburg, Germany
Haibo Hu	Hong Kong University of Science and Technology, China
Haixun Wang	IBM T.J. Watson Research Center, USA
Hakan Ferhatsmanoglu	Ohio State University, USA
Hayato Yamana	Waseda University, Japan
Heng Tao Shen	University of Queensland, Australia
H.V. Jagadish	University of Michigan, USA
Hyunchul Kang	Chung-Ang University, South Korea

Ibrahim Kamel	University of Sharjah, UAE
Indrakshi Ray	Colorado State University, USA
James Bailey	University of Melbourne, Australia
Jeffrey Xu Yu	Chinese University of Hong Kong, Hong Kong, China
Jiale Shen	University of Glasgow, UK
Jinyan Li	Institute for Infocomm Research, Singapore
Jun Miyazaki	Nara Institute of Science and Technology, Japan
Kamal Karlapalem	IIIT, Hyderabad, India
Katsumi Tanaka	Kyoto University, Japan
Keishi Tajima	Kyoto University, Japan
Kenji Hatano	Doshisha University, Japan
K. Selcuk Candan	Arizona State University, USA
Kazumasa Yokota	Okayama Prefectural University, Japan
Kazunari Ito	Aoyama Gakuin University, Japan
Kazutoshi Sumiya	University of Hyogo, Japan
Kyoji Kawagoe	Ritsumeikan University, Japan
Kyu-Young Whang	KAIST, Korea
Ladjel Bellatreche	LISI/ENSMA, France
Linhao Xu	National University of Singapore, Singapore
Li Yang	Western Michigan University, USA
Luc Bouganim	INRIA, France
Manolis Koubarakis	National and Kapodistrian University of Athens, Greece
Markus Schneider	University of Florida, USA
Masatoshi Arikawa	The University of Tokyo, Japan
Masayoshi Arisugi	Gunma University, Japan
Matthew Dailey	Asian Institute of Technology, Thailand
Md Maruf Hasan	Shinawatra University, Thailand
Miyuki Nakano	University of Tokyo, Japan
Mizuho Iwaihara	Kyoto University, Japan
Nandlal Sarda	Indian Institute of Technology Bombay, India
Oded Shmueli	Technion-Israel Institute of Technology, Israel
Ozgur Ulusoy	Bilkent University, Turkey
Panos Kalnis	National University of Singapore, Singapore
Photchanan Ratanajaipan	Shinawatra University, Thailand
Pierangela Samarati	Universita` degli Studi di Milano, Italy
Pongtawat Chippimolchai	Asian Institute of Technology, Thailand
P. Radha Krishna	Institute for Development and Research in Banking Technology, India
Qiankun Zhao	The Pennsylvania State University, USA
Rachada Kongkachandra	Thammasat University, Thailand
Rachanee Ungrangsi	Shinawatra University, Thailand
Rajugan R.	University of Technology, Sydney (UTS), Australia
Rui Zhang	University of Melbourne, Australia
Sanghyun Park	Yonsei University, Korea

Sanjay Madria	University of Missouri-Rolla, USA
Sean Wang	The University of Vermont, USA
Sengar Vibhuti Singh	Microsoft Research, India
Sergio Lifschitz	Pontifícia Universidade Católica do Rio de Janeiro, Brazil
Shuigeng Zhou	Fudan University, China
Shyam Kumar Gupta	IIT Delhi, India
Simonas Saltenis	Aalborg University, Denmark
Sonia Berman	University of Cape Town, South Africa
Sourav Bhowmick	Nanyang Technological University, Singapore
Sreenivasa Kumar P.	IIT Madras, India
Stefan Manegold	CWI, The Netherlands
Stephane Bressan	National University of Singapore, Singapore
Steven Gordon	Thammasat University, Thailand
Sujeet Pradhan	Kurashiki University of Science and the Arts, Japan
Sunil Prabhakar	Purdue University, USA
Sushil Jajodia	George Mason University, USA
Takahiro Hara	Osaka University, Japan
Takashi Tomii	Yokohama National University, Japan
Takeo Kunishima	Okayama Perfectural University, Japan
Takuya Maekawa	NTT, Japan
Thanaruk Theeramunkong	Sirindhorn International Institute of Technology, Thammasat University, Thailand
Thanwadee Sunetnanta	Mahidol University, Thailand
Theo Haerder	University of Kaiserslautern, Germany
Tore Risch	Uppsala University, Sweden
Toshiyuki Amagasa	University of Tsukuba, Japan
Vasilis Vassalos	Athens University of Economics and Business, Greece
Verayuth Lerntanatee	Silpakorn University, Thailand
Vicenc Torra	III-A-CSIC, Catalonia, Spain
Vijay Atluri	Rutgers University, USA
Wang-Chien Lee	Penn State University, USA
Weining Qian	Fudan University, P.R. China
Wei Wang	University of North Carolina at Chapel Hill, USA
Weiyi Meng	SUNY at Binghamton University, USA
Willem Jonker	Philips Research, The Netherlands
Wolfgang Nejdl	L3S and University of Hannover, Germany
Xiaofang Zhou	University of Queensland, Australia
Xiaofeng Meng	Renmin University of China, China
Xuemin Lin	University of New South Wales, Australia
Yanfeng Shu	The University of Queensland, Australia
Yang-Sae Moon	Kangwon National University, Korea
Yan Wang	Macquarie University, Australia
Yasuhiko Morimoto	Hiroshima University, Japan

Yasushi Sakurai	NTT, Japan
Ying Chen	IBM, China
Young-Koo Lee	Kyoung Hee University, Korea
Yufei Tao	Chinese University of Hong Kong, China

Industrial Program Committee

Arvind R Hulgeri	Persistent Systems, India
Katsumi Takahashi	NTT Lab, Japan
Ming Xiong	Lucent/Bell Labs, USA
Prasad M Deshpande	IBM Research, India
Yasuhiro Kanemasa	Fujitsu Lab, Japan

External Referees

Alex Liu	University of Texas, USA
Amit Garde	Persistent Systems, India
Chavdar Botev	Cornell University, USA
Fan Yang	Cornell University, USA
Feng Shao	Cornell University, USA
L. Venkata Subramaniam	IBM IRL, India
Man Lung Yiu	University of Hong Kong, China
Meghana Deodhar	IBM IRL, India
Mirek Riedewald	Cornell University, USA
Panagiotis Karras	University of Hong Kong, China
Pankaj Kankar	IBM IRL, India
R. Venkateswaran	Persistent Systems, India
Shipra Agrawal	Bell Labs Research, India
Sourashis Roy	IBM IRL, India
Suju Rajan	University of Texas, USA
Sunita Sarawagi	IIT Bombay, India
Umesh Bellur	IIT Bombay, India

External Reviewers

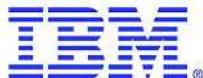
A. Balachandran	Persistent Systems, India
Amit Garde	Persistent Systems, India
Atsushi Kubota	Fujitsu Laboratories, Japan
Daniel Lieuwen	Bell Labs, USA
Deepak P	IBM Research, India
Iko Pramudiono	NTT, Japan
Krishna Kummamuru	IBM Research, India
Masanori Goto	Fujitsu Laboratories, Japan
Noriaki Kawamae	NTT, Japan
Nicolas Anciaux	INRIA, France
Nicolas Travers	University of Versailles, France
Philippe Pucheral	INRIA, France

R. Venkateswaran
Satyanarayana Valluri
Takeshi Motohashi
Toshikazu Ichikawa
Vijil E. Chenthamarakshan
Vinod G. Kulkarni

Persistent Systems, India
IIIT-Hyderabad, India
NTT, Japan
NTT, Japan
IBM Research, India
Persistent Systems, India

Sponsoring Institutions

IBM, Thailand



Korea Information Science Society



Database Society of Japan



National Electronics and Computer Technology Center



Software Industry Promotion Agency



Table of Contents

Invited Talks

‘Socio Sense’ and ‘Cyber Infrastructure for Information Explosion Era’: Projects in Japan	1
<i>Masaru Kitsuregawa</i>	
Is (Your) Database Research Having Impact?	3
<i>Guy M. Lohman</i>	

Part I: Full Papers

Query Language and Query Optimization - I

Improving Quality and Convergence of Genetic Query Optimizers	6
<i>Victor Muntés-Mulero, Néstor Lafón-Gracia, Josep Aguilar-Saborit, and Josep-L. Larriba-Pey</i>	
Cost-Based Query Optimization for Multi Reachability Joins	18
<i>Jiefeng Cheng, Jeffrey Xu Yu, and Bolin Ding</i>	
A Path-Based Approach for Efficient Structural Join with Not-Predicates.....	31
<i>Hanyu Li, Mong Li Lee, Wynne Hsu, and Ling Li</i>	

Query Language and Query Optimization - II

RRPJ: Result-Rate Based Progressive Relational Join	43
<i>Wee Hyong Tok, Stéphane Bressan, and Mong-Li Lee</i>	
GChord: Indexing for Multi-Attribute Query in P2P System with Low Maintenance Cost	55
<i>Minqi Zhou, Rong Zhang, Weineng Qian, and Aoying Zhou</i>	
ITREKS: Keyword Search over Relational Database by Indexing Tuple Relationship.....	67
<i>Jiang Zhan and Shan Wang</i>	

Data Mining and Knowledge Discovery

An MBR-Safe Transform for High-Dimensional MBRs in Similar Sequence Matching	79
<i>Yang-Sae Moon</i>	
Mining Closed Frequent Free Trees in Graph Databases	91
<i>Peixiang Zhao and Jeffrey Xu Yu</i>	
Mining Time-Delayed Associations from Discrete Event Datasets	103
<i>K.K. Loo and Ben Kao</i>	

Clustering

A Comparative Study of Ontology Based Term Similarity Measures on PubMed Document Clustering	115
<i>Xiaodan Zhang, Liping Jing, Xiaohua Hu, Michael Ng, and Xiaohua Zhou</i>	
An Adaptive and Efficient Unsupervised Shot Clustering Algorithm for Sports Video	127
<i>Jia Liao, Guoren Wang, Bo Zhang, Xiaofang Zhou, and Ge Yu</i>	
A Robust Feature Normalization Scheme and an Optimized Clustering Method for Anomaly-Based Intrusion Detection System	140
<i>Jungsuk Song, Hiroki Takakura, Yasuo Okabe, and Yongjin Kwon</i>	
Detection and Visualization of Subspace Cluster Hierarchies	152
<i>Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, Ina Müller-Gorman, and Arthur Zimek</i>	

Outlier Detection

Correlation-Based Detection of Attribute Outliers	164
<i>Judice L.Y. Koh, Mong Li Lee, Wynne Hsu, and Kai Tak Lam</i>	
An Efficient Histogram Method for Outlier Detection	176
<i>Matthew Gebski and Raymond K. Wong</i>	

Privacy Preserving Data Mining

Efficient k -Anonymization Using Clustering Techniques	188
<i>Ji-Won Byun, Ashish Kamra, Elisa Bertino, and Ninghui Li</i>	
Privacy Preserving Data Mining of Sequential Patterns for Network Traffic Data	201
<i>Seung-Woo Kim, Sanghyun Park, Jung-Im Won, and Sang-Wook Kim</i>	
Privacy Preserving Clustering for Multi-party	213
<i>Weijia Yang and Shanteng Huang</i>	
Privacy-Preserving Frequent Pattern Sharing	225
<i>Zhihui Wang, Wei Wang, Baile Shi, and S.H. Boey</i>	

Parallel and Distributed Databases

K_n Best - A Balanced Request Allocation Method for Distributed Information Systems	237
<i>Jorge-Arnulfo Quiané-Ruiz, Philippe Lamarre, and Patrick Valduriez</i>	
The Circular Two-Phase Commit Protocol	249
<i>Heine Kolltveit and Svein-Olaf Hvasshovd</i>	
Towards Timely ACID Transactions in DBMS	262
<i>Marco Vieira, António C. Costa, and Henrique Madeira</i>	

Data Warehouse

- BioDIFF: An Effective Fast Change Detection Algorithm for Biological Annotations *Yang Song, Sourav S. Bhowmick, and C. Forbes Dewey Jr.* 275

- An Efficient Implementation for MOLAP Basic Data Structure and Its Evaluation *K.M. Azharul Hasan, Tatsuo Tsuji, and Ken Higuchi* 288

Information Retrieval

- Monitoring Heterogeneous Nearest Neighbors for Moving Objects Considering Location-Independent Attributes *Yu-Chi Su, Yi-Hung Wu, and Arbee L.P. Chen* 300

- Similarity Joins of Text with Incomplete Information Formats *Shaouxu Song and Lei Chen* 313

- Self-tuning in Graph-Based Reference Disambiguation *Rabia Nuray-Turan, Dmitri V. Kalashnikov, and Sharad Mehrotra* 325

- Probabilistic Nearest-Neighbor Query on Uncertain Objects *Hans-Peter Kriegel, Peter Kunath, and Matthias Renz* 337

Indexing and Caching Databases

- Making the Most of Cache Groups *Andreas Bümann and Theo Härdter* 349

- Construction of Tree-Based Indexes for Level-Contiguous Buffering Support *Tomáš Skopal, David Hoksza, and Jaroslav Pokorný* 361

- A Workload-Driven Unit of Cache Replacement for Mid-Tier Database Caching *Xiaodan Wang, Tanu Malik, Randal Burns, Stratos Papadomanolakis, and Anastassia Ailamaki* 374

- J⁺-Tree: A New Index Structure in Main Memory *Hua Luan, Xiaoyong Du, Shan Wang, Yongzhi Ni, and Qiming Chen* 386

- CST-Trees: Cache Sensitive T-Trees *Ig-hoon Lee, Junho Shim, Sang-goo Lee, and Jonghoon Chun* 398

Security and Integrity Maintenance

- Specifying Access Control Policies on Data Streams *Barbara Carminati, Elena Ferrari, and Kian Lee Tan* 410

- Protecting Individual Information Against Inference Attacks in Data Publishing *Chen Li, Houtan Shirani-Mehr, and Xiaochun Yang* 422

- Quality Aware Privacy Protection for Location-Based Services *Zhen Xiao, Xiaofeng Meng, and Jianliang Xu* 434

Implementation of Bitmap Based Incognito and Performance Evaluation	447
<i>Hyun-Ho Kang, Jae-Myung Kim, Gap-Joo Na, and Sang-Won Lee</i>	
Prioritized Active Integrity Constraints for Database Maintenance	459
<i>Luciano Caroprese, Sergio Greco, and Cristian Molinaro</i>	
Image and Ontology-Based Databases	
Using Redundant Bit Vectors for Near-Duplicate Image Detection	472
<i>Jun Jie Foo and Ranjan Sinha</i>	
OLYBIA: Ontology-Based Automatic Image Annotation System Using Semantic Inference Rules	485
<i>Kyung-Wook Park, Jin-Woo Jeong, and Dong-Ho Lee</i>	
OntoDB: An Ontology-Based Database for Data Intensive Applications	497
<i>Hondjack Dehainsala, Guy Pierra, and Ladje Bellatreche</i>	
Sensor and Scientific Database Applications	
Continuously Maintaining Sliding Window Skylines in a Sensor Network	509
<i>Junchang Xin, Guoren Wang, Lei Chen, Xiaoyi Zhang, and Zhenhua Wang</i>	
Bayesian Reasoning for Sensor Group-Queries and Diagnosis	522
<i>Ankur Jain, Edward Y. Chang, and Yuan-Fang Wang</i>	
Telescope: Zooming to Interesting Skylines	539
<i>Jongwuk Lee, Gae-won You, and Seung-won Hwang</i>	
Eliciting Matters – Controlling Skyline Sizes by Incremental Integration of User Preferences	551
<i>Wolf-Tilo Balke, Ulrich Güntzer, and Christoph Lofi</i>	
Mobile Databases	
Optimizing Moving Queries over Moving Object Data Streams	563
<i>Dan Lin, Bin Cui, and Dongqing Yang</i>	
MIME: A Dynamic Index Scheme for Multi-dimensional Query in Mobile P2P Networks	576
<i>Ping Wang, Lidan Shou, Gang Chen, and Jinxiang Dong</i>	
Temporal and Spatial Databases	
Interval-Focused Similarity Search in Time Series Databases	586
<i>Johannes Aßfalg, Hans-Peter Kriegel, Peer Kröger, Peter Kunath, Alexey Pryakhin, and Matthias Renz</i>	
Adaptive Distance Measurement for Time Series Databases	598
<i>Van M. Chhieng and Raymond K. Wong</i>	

Clustering Moving Objects in Spatial Networks	611
<i>Jidong Chen, Caifeng Lai, Xiaofeng Meng, Jianliang Xu, and Haibo Hu</i>	

Data Streams

The Tornado Model: Uncertainty Model for Continuously Changing Data	624
<i>Byunggu Yu, Seon Ho Kim, Shayma Alkobaisi, Wan D. Bae, and Thomas Bailey</i>	
<i>ClusterSheddy: Load Shedding Using Moving Clusters over Spatio-temporal Data Streams</i>	637
<i>Rimma V. Nehme and Elke A. Rundensteiner</i>	
Evaluating MAX and MIN over Sliding Windows with Various Size Using the Exemplary Sketch	652
<i>Jiakui Zhao, Dongqing Yang, Bin Cui, Lijun Chen, and Jun Gao</i>	
CLAIM: An Efficient Method for Relaxed Frequent Closed Itemsets Mining over Stream Data	664
<i>Guojie Song, Dongqing Yang, Bin Cui, Baihua Zheng, Yunfeng Liu, and Kunqing Xie</i>	

P2P and Grid-Based Data Management

Capture Inference Attacks for K-Anonymity with Privacy Inference Logic	676
<i>Xiaojun Ye, Zude Li, and Yongnian Li</i>	
Schema Mapping in P2P Networks Based on Classification and Probing	688
<i>Guoliang Li, Beng Chin Ooi, Bei Yu, and Lizhu Zhou</i>	
ABIDE: A Bid-Based Economic Incentive Model for Enticing Non-cooperative Peers in Mobile-P2P Networks	703
<i>Anirban Mondal, Sanjay Kumar Madria, and Masaru Kitsuregawa</i>	

XML Databases

An Efficient Encoding and Labeling for Dynamic XML Data	715
<i>Jun-Ki Min, Jihyun Lee, and Chin-Wan Chung</i>	
An Original Semantics to Keyword Queries for XML Using Structural Patterns	727
<i>Dimitri Theodoratos and Xiaoying Wu</i>	
Lightweight Model Bases and Table-Driven Modeling	740
<i>Hung-chih Yang and D. Stott Parker</i>	

XML Indexing

An Efficient Index Lattice for XML Query Evaluation	753
<i>Wilfred Ng and James Cheng</i>	

A Development of Hash-Lookup Trees to Support Querying Streaming XML	768
<i>James Cheng and Wilfred Ng</i>	
Efficient Integration of Structure Indexes of XML	781
<i>Taro L. Saito and Shinichi Morishita</i>	
Efficient Support for Ordered XPath Processing in Tree-Unaware Commercial Relational Databases	793
<i>Boon-Siew Seah, Klarinda G. Widjanarko, Sourav S. Bhowmick, Byron Choi, and Erwin Leonardi</i>	

XML Query Processing

On Label Stream Partition for Efficient Holistic Twig Join	807
<i>Bo Chen, Tok Wang Ling, M. Tamer Özsu, and Zhenzhou Zhu</i>	
Efficient XML Query Processing in RDBMS Using GUI-Driven Prefetching in a Single-User Environment	819
<i>Sandeep Prakash, Sourav S. Bhowmick, Klarinda G. Widjanarko, and C. Forbes Dewey Jr.</i>	
Efficient Holistic Twig Joins in Leaf-to-Root Combining with Root-to-Leaf Way	834
<i>Guoliang Li, Jianhua Feng, Yong Zhang, and Lizhu Zhou</i>	
<i>TwigList</i> : Make Twig Pattern Matching Fast	850
<i>Lu Qin, Jeffrey Xu Yu, and Bolin Ding</i>	

Part II: Short Papers

Query Language and Query Optimization

CircularTrip: An Effective Algorithm for Continuous k NN Queries	863
<i>Muhammad Aamir Cheema, Yidong Yuan, and Xuemin Lin</i>	
Optimizing Multiple In-Network Aggregate Queries in Wireless Sensor Networks	870
<i>Huei-You Yang, Wen-Chih Peng, and Chia-Hao Lo</i>	
Visible Nearest Neighbor Queries	876
<i>Sarana Nutanong, Egemen Tanin, and Rui Zhang</i>	
On Query Processing Considering Energy Consumption for Broadcast Database Systems	884
<i>Shinya Kitajima, Jing Cai, Tsutomu Terada, Takahiro Hara, and Shojiro Nishio</i>	

Data Mining and Knowledge Discovery

Mining Vague Association Rules	891
<i>An Lu, Yiping Ke, James Cheng, and Wilfred Ng</i>	

An Optimized Process Neural Network Model	898
<i>Guojie Song, Dongqing Yang, Yunfeng Liu, Bin Cui, Ling Wu, and Kuning Xie</i>	
Clustering XML Documents Based on Structural Similarity	905
<i>Guangming Xing, Zhonghang Xia, and Jinhua Guo</i>	
The Multi-view Information Bottleneck Clustering	912
<i>Yan Gao, Shiwen Gu, Jianhua Li, and Zhining Liao</i>	
Web and Information Retrieval	
Web Service Composition Based on Message Schema Analysis	918
<i>Aiqiang Gao, Dongqing Yang, and Shiwei Tang</i>	
SQORE: A Framework for Semantic Query Based Ontology Retrieval...	924
<i>Chutiporn Anutariya, Rachanee Ungrangsi, and Vilas Wuwongse</i>	
Graph Structure of the Korea Web	930
<i>In Kyu Han, Sang Ho Lee, and Soowon Lee</i>	
EasyQuerier: A Keyword Based Interface for Web Database Integration System	936
<i>Xian Li, Weiyi Meng, and Xiaofeng Meng</i>	
Database Applications and Security	
Anomalies Detection in Mobile Network Management Data.....	943
<i>Marco Anisetti, Claudio A. Ardagna, Valerio Bellandi, Elisa Bernardoni, Ernesto Damiani, and Salvatore Reale</i>	
Security-Conscious XML Indexing.....	949
<i>Yan Xiao, Bo Luo, and Dongwon Lee</i>	
Framework for Extending RFID Events with Business Rule	955
<i>Mikyeong Moon, Seongjin Kim, Keunhyuk Yeom, and Heeseok Choi</i>	
Ontology and Data Streams	
Approximate Similarity Search over Multiple Stream Time Series	962
<i>Xiang Lian, Lei Chen, and Bin Wang</i>	
WT-Heuristics: A Heuristic Method for Efficient Operator Ordering	969
<i>Jun-Ki Min</i>	
An Efficient and Scalable Management of Ontology.....	975
<i>Myung-Jae Park, Jihyun Lee, Chun-Hee Lee, Jiexi Lin, Olivier Serres, and Chin-Wan Chung</i>	
Estimating Missing Data in Data Streams.....	981
<i>Nan Jiang and Le Gruenwald</i>	
XML Databases	
AB-Index: An Efficient Adaptive Index for Branching XML Queries	988
<i>Bo Zhang, Wei Wang, Xiaoling Wang, and Aoying Zhou</i>	

Semantic XPath Query Transformation: Opportunities and Performance	994
<i>Dung Xuan Thi Le, Stephane Bressan, David Taniar, and Wenny Rahayu</i>	
TGV: A Tree Graph View for Modeling Untyped XQuery	1001
<i>Nicolas Travers, Tuyêt Trâm Dang Ngoc, and Tianxiao Liu</i>	
Indexing Textual XML in P2P Networks Using Distributed Bloom Filters	1007
<i>Clement Jamard, Georges Gardarin, and Laurent Yeh</i>	
Towards Adaptive Information Merging Using Selected XML Fragments	1013
<i>Ho-Lam Lau and Wilfred Ng</i>	

Part III: Posters

Data Warehouse and Data Mining

LAPIN: Effective Sequential Pattern Mining Algorithms by Last Position Induction for Dense Databases	1020
<i>Zhenglu Yang, Yitong Wang, and Masaru Kitsuregawa</i>	
Spatial Clustering Based on Moving Distance in the Presence of Obstacles	1024
<i>Sang-Ho Park, Ju-Hong Lee, and Deok-Hwan Kim</i>	
Tracing Data Transformations: A Preliminary Report	1028
<i>Gang Qian and Yisheng Dong</i>	

Query Processing

QuickCN: A Combined Approach for Efficient Keyword Search over Databases	1032
<i>Jun Zhang, Zhaohui Peng, and Shan Wang</i>	
Adaptive Join Query Processing in Data Grids: Exploring Relation Partial Replicas and Load Balancing	1036
<i>Donghua Yang, Jianzhong Li, and Hong Gao</i>	

Efficient Semantically Equal Join on Strings	1041
<i>Juggapong Natwichai, Xingzhi Sun, and Maria E. Orlowska</i>	

Database Modeling and Information Retrieval

Integrating Similarity Retrieval and Skyline Exploration Via Relevance Feedback	1045
<i>Yiming Ma and Sharad Mehrotra</i>	
An Image-Semantic Ontological Framework for Large Image Databases	1050
<i>Xiaoyan Li, Lidan Shou, Gang Chen, and Kian-Lee Tan</i>	

Flexible Selection of Wavelet Coefficients for Continuous Data Stream Reduction	1054
Jaehoon Kim and Seog Park	
Versioned Relations: Support for Conditional Schema Changes and Schema Versioning	1058
Peter Sune Jørgensen and Michael Böhlen	
Network and XML Databases	
Compatibility Analysis and Mediation-Aided Composition for BPEL Services	1062
Wei Tan, Fangyan Rao, Yushun Fan, and Jun Zhu	
Expert Finding in a Social Network	1066
Jing Zhang, Jie Tang, and Juanzi Li	
Efficient Reasoning About XFDs with Pre-image Semantics	1070
Sven Hartmann, Sebastian Link, and Thu Trinh	
Part IV: Industrial Track	
Context RBAC/MAC Access Control for Ubiquitous Environment	1075
Kyu Il Kim, Hyuk Jin Ko, Hyun Sik Hwang, and Ung Mo Kim	
Extending PostgreSQL to Support Distributed/Heterogeneous Query Processing	1086
Rubao Lee and Minghong Zhou	
Geo-WDBMS: An Improved DBMS with the Function of Watermarking Geographical Data	1098
Min Huang, Xiang Zhou, Jiaheng Cao, and Zhiyong Peng	
Part V: Demonstrations Track	
TinTO: A Tool for the View-Based Analysis of Streams of Stock Market Data	1110
Andreas Behrend, Christian Dorau, and Rainer Manthey	
Danaïdes: Continuous and Progressive Complex Queries on RSS Feeds	1115
Wee Hyong Tok, Stéphane Bressan, and Mong-Li Lee	
OntoDB: It Is Time to Embed Your Domain Ontology in Your Database	1119
Stéphane Jean, Hondjack Dehainsala, Dung Nguyen Xuan, Guy Pierra, Ladjel Bellatreche, and Yamine Aït-Ameur	
Author Index	1123

‘Socio Sense’ and ‘Cyber Infrastructure for Information Explosion Era’: Projects in Japan

Masaru Kitsuregawa

University of Tokyo

kitsure@tkl.iis.u-tokyo.ac.jp

Some of the large projects in Japan where I am serving PI are introduced in this talk.

MEXT (Ministry of Education, Culture, Sports, Science and Technology) approved new project named ‘Cyber Infrastructure for Information Explosion Era’ in 2005. The year of 2005 was a preparation stage and we asked research proposals under this program. Totally seventy four research teams were accepted. The project effectively started on April 2006. This is the largest IT related project in the category of Grant-in-Aid for Scientific Research on Priority Areas. Around 5 million dollars for 2006. The project supposed to continue until FY2010. The amount of information created by people, generated by sensors and computers is explosively increasing recent years. Especially the growth ratio of web contents is very high. People do not ask questions to the friends anymore if they want to know something but use search engine and people are now really heavily dependent on the web. Knowledge workers are using a lot of time just for ‘search’. The more the information be generated, the more we find difficulty to locate appropriate information. In order to achieve higher quality search, we are currently developing an open next generation search engine incorporating deep NLP capabilities. By deep, we mean we put more machine power to web contents analysis. In another words, we do not care about response time, since current 1 sec response time is dependent on the advertisement based monetization scheme. We believe we should provide service, which is more than ordinary search. In addition to web, we do have yet another information explosion in the area so called e-science. Through introduction of very powerful supercomputer and various kinds of advanced sensor systems, science is now becoming very data intensive. We plan to build tools for science discovery over the sea of data explosion. Another area would be health care. A lot of patient health care records are now becoming to be digitally stored. Monitoring the human activities with sensors and mining the archived HCR would be typical data driven application.

Explosion of the information incurs several problems not just in search but also in computer system management. A lot of information means a lot of applications, which gives so much stresses against the system. Cost of maintaining the system is now increasing more and more. Self monitoring the system activities also generate huge amount of information. BAM is one typical higher level example. We are now building large scale distributed cluster test bed over Japan, which is a shared platform for next generation system software development.

Human interaction is also very important research issue. All the information finally has to be absorbed by people. Highly functional human interaction capturing room are being developed. Various kinds of sensors are prepared and eight video

cameras capture the interaction process in synchronously from different angles. Building the interaction corpus would be important for several modal analysis researches.

Thus information explosion project covers almost all the computer science areas. More than 200 researchers are now participating.

Socio-sense project will be also introduced. People are spending more time in the cyber world in addition to in the real world. Most of the important events are immediately reflected onto the cyber world, which means we can capture the activities of the real world through the cyber world, whose information can be crunched by information technology. Cyber world can be regarded as a SENSOR for the real world. By viewing the evolution of cyber world, we can interpret various interesting social activities. The system of socio sense is not a search engine but a kind of tool to see the societal behavior. This is also supported by MEXT.

METI is going to start 'Grand Information Voyage' project from April 2007.

Several national projects on information explosion starts(ed) in Japan. We are considering the possibilities of international collaboration.

Information explosion project:

<http://itkaken.ex.nii.ac.jp/i-explosion/ctr.php/m/IndexEng/a/Index/>

Consortium for Grand Information Voyage project:

<http://www.jyouhoudaikoukai-consortium.jp/>

Is (Your) Database Research Having Impact?

Guy M. Lohman

IBM Almaden Research Center
650 Harry Rd., San Jose, CA 95120
lohman@almaden.ibm.com

Is your research having real impact? The ultimate test of the research done by this community is how it impacts society. Perhaps the most important metric of this impact is acceptance in the marketplace, i.e. incorporation into products that bring value to the purchaser. Merely publishing papers and getting them referenced has no intrinsic value unless the ideas therein are eventually used by someone. So let us ask ourselves candidly – is (my) database research having (positive) impact? Concisely: Are they buying my stuff? Have the “hot topics” of the past withstood the test of time by actually being used in products that sold? If so, what characteristics were instrumental in their success? And if not, why did something that got so many people excited fail to gain traction with users? Perhaps more importantly, what can we learn from our track record of the past in order to have better impact in the future? How can we better serve our user community by solving their real problems, not the ones we may imagine?

Let us first **critique** our historical track record as a community. Over the last thirty years, a few major topics seem to have dominated the interest of the research community in databases. Waves of “hot topics” appear to rise to predominance, in terms of the number of papers submitted (and hence published), and after a few years of excitement get replaced by another topic. Not that these waves exclude other topics or are cleanly delineated – they simply seem to coincidentally interest a large proportion of our community. I will not attempt to justify this premise with statistics on topics; it’s just an observation that many experienced researchers recognize. The first of these with which I’m familiar was relational databases, themselves, which captivated the attention of database researchers in the last half of the 1970s, resulting in major prototypes such as System R, Ingres, and others that formed the foundation of products in the early 1980s. Distributed databases seemed to dominate the early 1980s, but this thread rapidly evolved into separate threads on parallel databases and the integration of disjoint (and often heterogeneous) databases, usually called federated databases. In the late 1980s and early 1990s, object-oriented databases attempted to address the requirements of some under-served applications, and the relational crowd fought back by creating “extensible” databases with “object-relational” extensions to meet the OODBMS challenge. About the same time, interest in Datalog created strong interest in deductive databases. The mid- and late-1990s saw the birth and explosion of interest in data warehousing and data mining, eventually spawning a whole new research community in knowledge discovery. Around 1999, standardization of XML rocketed XML databases into the forefront. The early- to mid-2000s have seen great interest in streams and sensor databases. And along the way, numerous other variations on these themes have enjoyed the

spotlight for a while: database machines, temporal databases, multi-media and spatial databases, scientific and statistical databases, active databases, semantic databases and knowledge bases, and a recent favorite of mine that has yet to gain much interest from academia – self-managing databases. To what extent has each of these topics successfully impacted the marketplace, and why? We must learn from our successes and failures by carefully examining why the market accepted or rejected our technology.

My assessment is that our success has depended upon the *consumability* of our technology: how well it meets a customer need, how simple it is to understand and use, and how well standardization has stabilized its acceptance across vendors. Relational technology succeeded and has grown spectacularly to become a U.S. \$14 Billion industry in 2004 largely because it was simpler and easier to understand than its predecessors, with a declarative query language (SQL) that simplified application development, and was standardized early in its (product) evolution. However, attempts to “augment” it with object-relational, temporal, and deductive extensions have been either: (a) too complicated, (b) insufficiently vital to most consumers’ applications, and/or (c) not standardized or standardized too late in its evolution. Parallel databases exploited increasingly inexpensive hardware to facilitate growth and performance requirements with generally acceptable increases in complexity (mostly in administration, not querying), whereas federated databases have seen less success because the complexity of integrating diverse data sources largely fell on the user. Data mining, while a genuine success in the research community, evoked a comparative yawn in the marketplace largely because users needed to understand it to use it, and they had difficulty understanding it because of its novelty and mathematical intricacies. The jury is still out on XML databases, but my fear is that, despite the need for storing increasing volumes of XML data, XQuery is far more complicated than SQL. Similarly, stream databases are too new to be judged adequately, but I question the market size and whether the research in the database community adequately suits the “lean and mean” real-time requirements of the primary market – the investment and banking industries.

How then should we increase the impact of our research in the future? First, we must candidly assess our strengths and weaknesses. Our strengths lie in modeling the semantics underlying information, enabling better precision in our queries than the keyword search upon which Information Retrieval and the popular search engines are based. We have much to offer the IR and search communities here, and they have recognized this by aggressively hiring from the database community in the last few years. Our models also permit reasoning about the data through complex OLAP-style queries to extract actionable information from a sea of data. We know how to optimize a declarative language, and how to exploit massive parallelism, far better than any other discipline. Our primary weakness is in simplicity / usability, particularly in setting up and administering databases. This is certainly exacerbated by database researchers not gaining firsthand experience by routinely using databases to store their own data. Secondly, we must reach out to other disciplines with complementary strengths, and learn from them. Despite the lack of precision of keyword search, why is it vastly preferred over SQL? Third, we must engage with real users (which should include ourselves) and listen carefully to what they say. Have you ever tried to query or manage a non-trivial database of at least 500 tables

that was not constructed by you? Have you ever tried to add disks or nodes to an existing database that exceeded its initial space allocation? Have you ever built and administered a real application using a database? Did your research remedy any of the pain points you encountered or heard from a user? Fourth, we must go back to basics and design our systems based upon user requirements, not upon what technology we understand or want to develop.

Pursuing the fourth item in greater detail, we should honestly ask ourselves why less than 20% of the world's data is stored in databases. Weren't object-relational extensions supposed to rectify this by enabling storage of unstructured and semi-structured data, as well as structured data? Currently, users rely upon content managers to manage this unstructured and semi-structured content. Though content managers are built upon relational DBMSs, the content is stored in files, so isn't easily searched, and the search interface isn't SQL. This certainly isn't what users want. Users want a single, uniform interface to all their data, particularly for searching. Increasingly, they recognize that the majority of their costs are for people and their skills, as hardware costs are driven downward by Moore's Law. So lowering the Total Cost of Ownership (TCO) requires systems that are easier to manage and require fewer skilled people to manage. Users also want a scalable solution that permits easily adding more capacity to either the storage or the computing power in an incremental fashion as their needs for information management increase. The increasing requirements for compliance with government regulations, as well as business imperatives to extract more value out of information already collected in diverse application "silos", are driving their need to integrate systems never designed to interact with other systems, and to be able to more proactively and quickly derive business intelligence than with today's data warehouses. Ultimately, users want to be able to quickly and easily find, integrate, and aggregate the data that they need to make business decisions. But that data is currently scattered throughout their enterprise in a staggering array of incompatible systems, in a daunting tangle of differing formats. The usual lament is that they know the data is out there somewhere, but they can't find it.

Clearly there are plenty of hard research problems – as well as business opportunities! – in all of these requirements! We simply have to listen and be willing to change our research agendas to the problems that matter most to our "customers". And focusing on customer pain points doesn't preclude attempting risky, imaginative, cool, technically advanced, and occasionally far-out technical approaches. In fact, problems having origins in reality tend to be the most challenging. Only by doing so will our research withstand the test of time in the marketplace of ideas, and truly have the impact we all want for our work.

Improving Quality and Convergence of Genetic Query Optimizers*

Victor Muntés-Mulero¹, Néstor Lafón-Gracia¹, Josep Aguilar-Saborit², and Josep-L. Larriba-Pey¹

¹ DAMA-UPC, Computer Architecture Dept., Universitat Politècnica de Catalunya,
Campus Nord UPC, C/Jordi Girona Mòdul D6 Despatx 117 08034 Barcelona, Spain
`{vmuntes,nlafon,larri}@ac.upc.edu`,
<http://www.dama.upc.edu>

² IBM Canada Ltd. IBM Toronto Lab. 8200 Warden Ave. Markham, Ontario.
Canada L6G1C7
`jaguilar@ca.ibm.com`

Abstract. The application of genetic programming strategies to query optimization has been proposed as a feasible way to solve the large join query problem. However, previous literature shows that the potentiality of evolutionary strategies has not been completely exploited in terms of convergence and quality of the returned query execution plans (QEP).

In this paper, we propose two alternatives to improve the performance of a genetic optimizer and the quality of the resulting QEPs. First, we present a new method called *Weighted Election* that proposes a criterion to choose the QEPs to be crossed and mutated during the optimization time. Second, we show that the use of heuristics in order to create the initial population benefits the speed of convergence and the quality of the results. Moreover, we show that the combination of both proposals outperforms previous randomized algorithms, in the best cases, by several orders of magnitude for very large join queries.

1 Introduction

Query optimization based on evolutionary approaches is still an intriguing alternative to solve the very large join query problem. Advanced applications such as SAP or those involving information integration often need to combine a large set of tables to reconstruct complex business objects. For instance, the SAP schema may contain more than 10,000 relations [6] and may join more than 20 of these in a single SQL query. As the number of relations involved in a SQL statement increases, traditional optimizers, which are usually based on dynamic programming techniques [13], fail to perform satisfactorily. The main problem

* Research supported by the IBM Toronto Lab Centre for Advanced Studies and UPC Barcelona. The authors from DAMA-UPC want to thank Generalitat de Catalunya for its support through grant number GRE-00352 and Ministerio de Educacin y Ciencia of Spain for its support through grant TIN2006-15536-C02-02.

lies in the size of the search space, which grows exponentially with the increase of the number of relations involved in the query. In this scenario, users, or even the DBMS [20], are usually forced to split the query in smaller subqueries in order to optimize it, obtaining QEPs that are typically far from the optimum.

Genetic approaches have proven to be a good alternative since they are included among the best randomized approaches in terms of quality and speed of convergence [15]. However, there are still important aspects to be studied in order to improve the performance of genetic approaches. On the one hand, evolutionary algorithms perform a beam search, based on the evolution of a population, instead of focusing on the evolution of a single individual [1], as opposed to random-walk algorithms like *iterative improvement* or *simulated annealing*. Although this can be beneficial in terms of quality, it may jeopardize the ability of the optimizer to converge quickly. On the other hand, recent studies show, by means of a statistical model, that the random effects of the initial population cannot be neglected, since they have a significant impact on the quality of the returned QEP after the optimization process [11]. In other words, depending on the small sample of QEPs created at random for the initial population, the genetic optimizer will experience difficulties to find a near optimal QEP. This is aggravated by the fact that the search space grows exponentially as the number of relations increases, which implies that the size of the initial population should also grow exponentially.

In order to remedy these two drawbacks, we propose two different approaches. We call our first proposal *Weighted Election* (WE) and it tackles the problem of the speed of convergence mentioned above. In all the traditional evolutionary algorithms, the members of the population chosen to be crossed with other members or mutated are chosen at random. WE proposes a new approach where the QEPs are chosen with a certain probability depending on their associated cost, giving more probability to low-costed plans to be chosen as opposed to high-costed plans. Our second approach is aimed at reducing the variability in the quality of the results, introduced by the random effects of the initial population, by using heuristics to assure that the first sample of QEPs is not blindly chosen from the search space, but it follows a minimum quality criterion. We call this approach *Heuristic Initial Population* (HIP).

Finally, we show that the combination of both approaches is beneficial. Specifically, we compare our new approach with the Two-Phase Optimization algorithm (2PO) [4], which is considered to be the best randomized algorithm presented in the literature. We show that our techniques significantly improve a genetic optimizer and, in addition, are more suitable than previous randomized techniques for very large join queries.

This paper is organized as follows. Section 2 introduces genetic optimization and the genetic optimizer used in this work. Section 3 and 4 describe our proposals in detail. In Section 5, we present the results obtained by the comparison of the different algorithms. Finally, in Sections 6 and 7, we present related work and conclude.

2 The Carquinyoli Genetic Optimizer (CGO)

The Carquinyoli Genetic Optimizer (CGO) is, to the best of our knowledge, the most sophisticated genetic approach presented in the literature and the first genetic optimizer tested against a well-known commercial optimizer [8]. For this reason we use CGO as the baseline of our work.

CGO is based on genetic programming. Inspired by the principles of natural selection, the basic idea of genetic programming is, given an initial set of *programs*, generally called members of an initial population, to perform a set of operations in order to get a well-fitted program able to solve a specific task. Each member or program in the population represents a way to achieve a specific objective and has an associated cost.

Starting with this initial population, usually created from scratch, two operations are used to produce new members in the population: (i) *crossover operations*, which combine properties of two members in the population chosen at random, and (ii) *mutation operations*, which introduce new properties into a randomly chosen member in the population. In order to keep the size of the population constant, a third operation, usually referred to as *selection*, is used to discard the worst fitted members, using a fitness function. This process generates a new population, also called *generation*, that includes both the old and the new members that have survived to the selection operation. This is repeated iteratively until a *stop condition* ends the execution. Once the stop criterion is met, the best solution is taken from the final population. Query optimization can be reduced to a search problem where the DBMS needs to find the optimum query execution plan (QEP) in a vast search space. Each QEP can be considered as a possible solution (or program) for the problem of finding a good access path to retrieve the required data. Therefore, in a genetic query optimizer, every member of the population is a QEP. Further details of CGO can be found in [8].

3 Weighted Election (WE)

Among the randomized algorithms, two different classes of algorithms have been applied to query optimization. On the one hand, we have the random-walk based algorithms, typically represented by *iterative improvement* and *simulated annealing* and all the improvements and combinations of these two, such as 2PO. On the other hand, there are proposals in the literature for the use of evolutionary techniques as an alternative way to achieve a near-optimal QEP. There is a fundamental difference between both alternatives: the philosophy of the search space is different. While random-walk algorithms rely on a single individual (QEP) and a sequence of transformations on this individual, evolutionary algorithms apply the transformations on a population. As a consequence, while genetic approaches keep more information than random-walk algorithms, which may lead the optimizer to find better-costed QEPs, they might experience a lower speed of convergence. This is sustained by the fact that they do not only keep the best QEP, but they spend some time optimizing QEPs that are not close to the optimal plan.

In order to improve this drawback of evolutionary strategies, we propose and analyze a new technique called Weighted Election (WE). This technique aims at directing the search towards the directions marked by the best QEPs in the population, by giving more opportunities to these QEPs to be crossed and mutated than QEPs with higher costs. Note that high-costed QEPs still have an opportunity to participate in the genetic operations performed by the optimizer, although the probability is lower.

In order to assign the weight of a QEP p in the population Pop , denoted by W_p , we use the following formula:

$$W_p = \max \left(\frac{\mu_{\frac{1}{2}} - C_p}{\mu_{\frac{1}{2}} - B_{Pop}} \cdot \alpha, 1 \right) \quad (1)$$

where C_p is the cost associated with QEP p , $\mu_{\frac{1}{2}}$ is the median cost in the population and B_{Pop} is the best cost in the population. Note that W_p ranges from 1 to α , where $\alpha > 1$. Specifically, QEPs with costs lower than the median are assigned a weight from 1 to α , while QEPs with costs higher than the median are assigned a cost of 1. Depending on the value of α we can give more or less importance to the differences between the costs of the QEPs in the population. For example, for $\alpha = 2$ and $\alpha = 100$, the probability of the QEP with the lowest cost in the population to be chosen is 2 and 100 times the probability of the highest-costed QEP, respectively.

4 Heuristic Initial Population (HIP)

The quality of the initial population can be decisive in order to obtain near-optimal QEPs. Unfortunately, since the initial population is usually created at random [15], its affect on the quality of the results is unpredictable. Our proposal assures that the quality of the initial population is higher than a randomly created population, using heuristics to create part of the plans in it.

Several heuristic algorithms have been proposed in the literature aiming at solving the query optimization problem. Representatives of this class of algorithms are the KBZ algorithm [7], the AB algorithm [19], the *Augmentation* algorithm (AG), and other greedy algorithms [14][15][17].

Because of its working principle, the KBZ algorithm requires the assignment of join implementations to join graph edges before the optimization is carried out. This requirement and the restrictions concerning the cost model do not allow the algorithm to approximate the real solution, when it deals with a sophisticated and detailed cost model [15]. AB was developed in order to solve the restrictions imposed by KBZ on the join implementation placement. However, even with the AB extension it is difficult to make use of a complex cost model.

The *Augmentation* algorithm (AG) is an incremental heuristic method to build QEPs. Specifically, 5 different criteria are studied, namely, choosing the relation with minimum cardinality, choosing the relation participating in the largest number of joins, choosing the joins with minimum selectivity, choosing an operation using the combination of the first and the third criteria and, finally,

using the so-called *KBZ rank*, related to the KBZ algorithm. Among the five criteria, the minimum selectivity criterion turned out to be the most efficient and, for this reason, it is the one selected for this work. Depending on the relation chosen to start the optimization process, different QEPs can be generated. In general, we consider that the AG algorithm does not generate a single QEP, but as many QEPs as relations involved in the query.

Algorithm 1. HIP: Initial Population generation pseudocode

```

1: procedure INIPOP(time maxTime, int maxPlans)
2:   int numPlan  $\leftarrow 0$ ;
3:   p  $\leftarrow$  generateMinimumJoinSelectivityPlan();
4:   currentTime  $\leftarrow$  getCurrentTime();
5:   while (p  $\wedge$  currentTime  $< \frac{\text{maxTime}}{2}$   $\wedge$  numPlan  $<$  maxPlans) do
6:     insertPlanToPopulation(p);
7:     numPlan  $\leftarrow$  numPlan + 1;
8:     p  $\leftarrow$  generateMinimumJoinSelectivityPlan();
9:     currentTime  $\leftarrow$  getCurrentTime();
10:    end while
11:   if (numPlans  $<$  maxPlans) then
12:     genRemainingRandomMembers(maxPlans - numPlans);
13:   end if
14: end procedure

```

Algorithm 1 summarizes the working principles of HIP. In order to simplify the implementation and the experiments, we assume that we fix the optimization time *a priori*. This optimization time is passed to Algorithm 1 using the parameter *maxTime*. A number of QEPs are created (lines 3 and 8) and introduced in the population (line 6) using the Minimum Join Selectivity heuristic (MJS). Since MJS has a non-trivial computational cost, generating all the members of the population with the heuristic could be very time-consuming, exhausting the whole optimization time, and preventing the genetic optimizer from performing an operation. Therefore, as shown in line 5, the heuristic is applied until the maximum number of possible QEPs generated by the heuristic is reached. Thus, we create as many QEPs as needed in the population or we spend about half of the optimization time. Finally, if the population is not completed after the loop, the remaining QEPs are created at random using the function *genRemainingRandomMembers()*. This function has a parameter that specifies the number of remaining QEPs to be created at random (line 12).

5 Experimental Results

Our first concern is to provide means to assure a fair comparison between the approaches studied in this paper. With this purpose, we have used the meta-structures created for CGO in order to implement the new techniques and 2PO,

i.e., the QEP metadata, the functions to calculate the cost of a plan, etc. With this, we guarantee that the efforts put on the performance optimization of CGO are also used by the other approaches.

Our new techniques are tested first with star schemas and for random queries, since they represent one of the most typical scenarios in *Decision Support Systems*, similar to those used for TPC-H. In order to provide means to generalize our conclusions, we also test our techniques with random queries. We do not show the results using the TPC-H benchmark since the number of relations in this schema does not allow the creation of large join queries.

Star Join Queries. For star join queries [3] we have randomly generated two databases containing 20 and 50 relations. Both schemas contain a large *fact table* or *central relation* and 19 and 49 smaller *dimension tables*, respectively. The fact table contains a foreign key attribute to a primary key in each dimension relation. We have distributed the cardinalities in order to have most of the dimensions with a significantly lower cardinality compared to the fact table. A few set of dimensions would have cardinalities closer to the cardinality of this fact table, but still at least one order of magnitude smaller, which typically corresponds to real scenarios (similar to the TPC-H database schema). The number of attributes per dimension, other than those included in the primary key, ranges from 1 to 10. The exact number of attributes per dimension and the attribute type is chosen at random. We define an index for every primary key.

We randomly define two sets of 9 star join queries, Q_{20} and Q_{50} , one for each database schema. Each set contains queries involving 20 and 50 relations, respectively. Every query includes all the relations of its corresponding database schema with at least one explicit join condition associated with each relation. Therefore, since CGO avoids cross products, we ensure that our queries are well defined star join queries.

Let Q be a SQL statement reading from a set of relations and γ the set of constraints in Q . Every constraint c in γ has an associated selectivity factor $s(c)$. In a star join query, every dimension table typically adds some information to the data flow or, if a constraint is affecting one of its attributes, it acts as a filter to discard those results not matching the constraint. Let us define S as the selectivity of the query calculated as $S = \prod_{c \in \gamma} s(c)$. Each set of queries Q_{20} and Q_{50} contains 9 queries $q_i, i = 1..9$ and, in both cases, $S(q_1) = S(q_2) = S(q_3) \approx 10^{-2}$, $S(q_4) = S(q_5) = S(q_6) \approx 10^{-4}$ and $S(q_7) = S(q_8) = S(q_9) \approx 10^{-8}$.

Random Queries. We have generated 30 random queries to evaluate our proposal. The set of random queries is divided into three groups involving 20, 50 and 100 join operations, respectively. In order to generate random queries we use two tools that we have created and called *rdbgen* and *rqgen*, described in [9].

Execution details. Every algorithm has been tested on all the queries. For each star join query, we have created 5 populations. Each population is used by all the algorithms, except for HIP, which creates a different initial population. This way, we eliminate possible noise relative to the random effects of the initial population and perform a fairer comparison. Every test on every evolutionary algorithm and

population consists of 10 executions each. We also run 10 executions for 2PO. In total, we have run 5280 executions. The experiments have been run on an Intel® Xeon® processor at 2.8 GHz with 2 GB of RAM. Either for evolutionary algorithms or 2PO, we use the *scaled cost* to compare results:

$$ScaledCost = \begin{cases} C_{orig}/C_{Tech} - 1 & \text{if } C_{orig} \geq C_{Tech} \\ 1 - C_{Tech}/C_{orig} & \text{if } C_{orig} < C_{Tech} \end{cases} \quad (2)$$

where C_{orig} represents the best cost obtained by the original implementation of CGO and C_{Tech} represents the best cost achieved by the specified technique to be tested. This way, the scaled cost in formula (2) allows us to obtain the average from the execution of different queries and databases and it is centered in 0. So if a technique has a positive scaled cost sc ($sc > 0$), it obtains QEPs with costs that are, on average, more than sc times lower than those obtained by CGO. A negative value indicates that the QEP obtained by that technique is, on average, worse than those obtained by CGO. From here on, we compare the techniques analyzed in this paper to CGO using formula (2).

Carquinyoli Genetic Optimizer (CGO). In order to parameterize CGO we use the recommendations obtained by the statistical analysis presented in [11]. Table I summarizes the values used to configure CGO.

Two-Phase Optimization (2PO). We have parameterized 2PO using the configuration proposed in [4]. During the first phase of 2PO, we perform 10 local optimizations using iterative improvement. The best QEP obtained in this first phase is used as the starting point, in the second phase, for the simulated annealing algorithm. The starting value for the initial temperature is the 10% of the cost of this QEP. The same parametrization for 2PO was also used in [15].

5.1 Weighted Election Analysis

As explained before, the difference between the probability to choose the best and the probability to choose the worst QEP in the population can be magnified depending on the value of parameter α . In order to study the effect of this parameter, each run is tested using five different values for α : 2, 10, 10^2 , 10^3 and 10^4 . We run our experiments using the two different sets of queries mentioned above, namely the star join query set, executing all the policies 10 times per each of the 5 populations created per query, and 30 random queries, where each policy is also run 10 times per configuration, in order to obtain averages.

Table 1. Parameters set used depending on the number of relations in the query. The number of crossover and mutation operations presented is executed per generation.

PARAMETER	# members	# cross	# mut
# Relations	20 50 100	20 50 100	20 50 100
Value	160 400 800	80 200 300	50 100 150

Figure II shows the results obtained after these experiments. The uppermost row shows the behavior of WE for star join queries involving 20 relations. The leftmost plot (plot *a*) corresponds to the star join queries with highest selectivity, i.e., those queries that return a larger number of matches ($S \approx 10^{-2}$). The plot in the middle (plot *b*) corresponds to queries with $S \approx 10^{-4}$ and the rightmost plot (plot *c*) to queries with lowest selectivity $S \approx 10^{-8}$. Since the number of relations is relatively small, close to what can still be handled by dynamic programming techniques, there is still little room for improvement. In general, the larger the value of α , the more significant the improvements introduced by WE. However, the plots show that the difference between $\alpha = 1000$ and $\alpha = 10000$ is not significant. We can also observe that, for very low selectivity, the gains of WE are reduced (plot *c*). This effect is explained by the fact that, when the selectivity is very small, most of the potential tuple results are discarded, resulting in a very low data flow cardinality in the QEP. Since the join operations can be executed in memory and do not incur extra I/O, all the QEPs have a similar cost and most of the executions of CGO are likely to reach a QEP with a near-optimal cost, reducing the chances for good performance.

Analogously, the central row of plots shows the same results for star join queries involving 50 relations. Our first observation is that, in some cases the gains obtained by WE are several orders of magnitude larger than those obtained by CGO. Again, we can observe that the general trend is to reward large values of

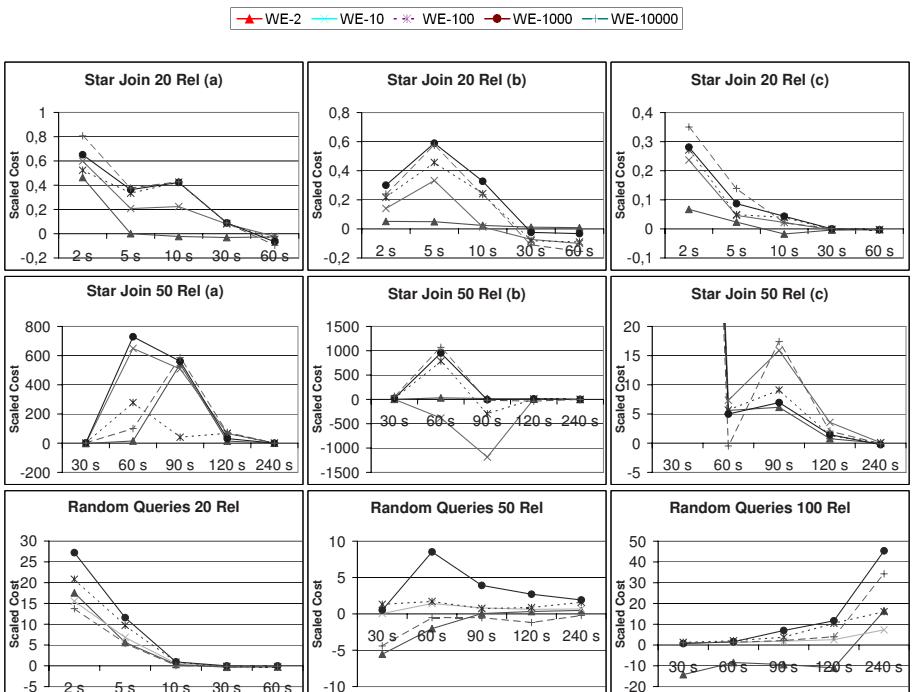


Fig. 1. Scaled Cost evolution for different values of α and different configurations

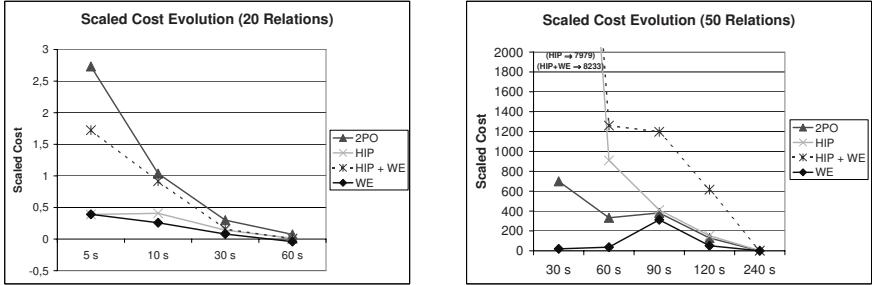


Fig. 2. Scaled Cost evolution for WE using $\alpha = 1000$, HIP, the combination of both and 2PO studying different number of relations for star join queries

α with better performance. Also, we would prefer the performance achieved for $\alpha = 1000$ instead of that achieved for $\alpha = 10000$, which is not as stable in all the situations. There is a trade-off for parameter α : it is recommendable to use larger values to achieve good performance (i.e., larger than 100), but too large values increase the probability of the best plan in the population to be chosen in such a way that, in practice, we are almost forcing the exclusive use of the best QEPs in the population, destroying one of the main differences between the genetic approaches and the random-walk approaches. Similarly, the improvements of WE decrease as the selectivity decreases for the reason explained above. However, in the worst cases we still obtain QEPs with costs that, in general, are several times larger than those obtained by CGO.

Finally, for random queries, in the lowermost row of plots, we observe the same trends as with the star join queries. Again, the best value of α tested is 1000, independently of the number of relations involved in the query. Extreme cases like $\alpha = 2$ or $\alpha = 10000$ must be avoided since they might lead to performances worst than those by CGO.

5.2 Heuristic Initial Population Analysis

In this section we analyze the benefits obtained by generating part of the population using HIP. Specifically, we run the same number of executions as in the previous analysis, using the same configurations. Figures 2 and 3 show the results of our analysis of this technique, and also the results described in the next subsection.

We first study the behavior for star join queries. In general, the use of HIP does always improve the performance of CGO. As suggested in [11], spending extra time generating good initial plans is clearly beneficial. Similar to what happens with WE, the improvements are in general very limited in the case of star join queries with 20 relations (left plot in Figure 2), since the search space has not grown enough to obtain QEPs that clearly differ, in terms of quality, from those obtained by CGO. However, for 50 relations (right plot in Figure 2) HIP obtains results that are three orders of magnitude better than those obtained by CGO. As the plot shows, for small optimization times, the improvement of our

techniques is around 10 times better than 2PO. It takes CGO about 4 minutes to achieve results similar to those generated by HIP, which implies that HIP converges much faster without losing quality. For random queries (Figure 3), we can observe that HIP also obtains results similar to those obtained for star join queries achieving, for queries containing 100 joins, improvement of more than four orders of magnitude.

5.3 Combining WE and HIP vs. 2PO

Finally, we combine both techniques and compare their behavior with the best random-walk algorithm presented in the literature: 2PO. All the experiments in this subsection have been run using $\alpha = 1000$. As it can be observed in Figures 2 and 3, the combination of HIP and WE in CGO clearly outperforms 2PO with star join and random queries, except for the case of 20 relations, where they behave very similarly. The benefits obtained by the combination of the two techniques presented in this paper obtain QEPs that are, on average, 20 times better than those obtained by 2PO, with 50 joins, and four orders of magnitude better for 100 joins. These results show that 2PO can be used as an intermediate solution for queries with about 20 joins, but it quickly fails to find QEPs for very large join queries, since the search space expands exponentially, and the random-walk algorithms potentiality degraded.

6 Related Work

The first approaches that applied genetic algorithms to query optimization considered a reduced set of QEP properties in crossover and mutation operations [2][15]. In these first proposals, the amount of information per plan is very limited because plans are transformed to chromosomes, represented as strings of integers. This lack of information usually leads to the generation of invalid plans that have to be repaired. In [16], a genetic-programming-based optimizer is proposed that directly uses QEPs as the members in the population, instead of using chromosomes. A first genetic optimizer prototype was created for PostgreSQL [12], but its search domain is reduced to left-deep trees and mutation operations are deprecated, thus bounding the search to only those properties appearing in the QEPs of the initial population. Besides, execution plans are

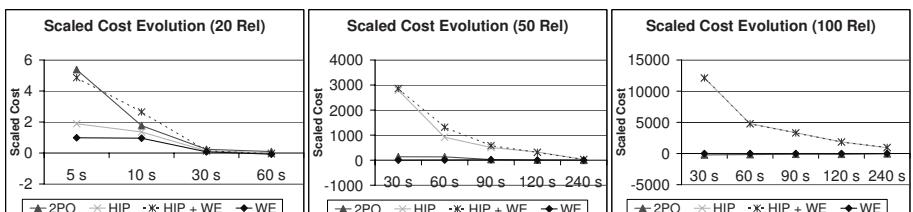


Fig. 3. Scaled Cost evolution for WE using $\alpha = 1000$, HIP, the combination of both and 2PO studying different numbers of relations for random queries

represented as strings of integers, thereby losing a lot of important information. CGO is presented in [8] and later analyzed in [10][11] showing that it is possible to find criteria to parameterize a genetic optimizer for star-join queries. Also, several variants of *random-walk* algorithms have been proposed in [4][5][15][18]. Randomized search techniques try to remedy the exponential explosion of dynamic programming techniques by iteratively exploring the search space and converging to a nearly optimal solution.

7 Conclusions

In this paper we present two techniques, namely *Weighted Election* (WE) and *Heuristic Initial Population* (HIP). These techniques tackle two important aspects of genetic optimization: the time wasted optimizing some QEPs in the population with a large cost and the effects of the initial population on the quality of the best QEP generated by the optimizer. WE is able to speed up a genetic optimizer and achieve a quick convergence compared to the original, meaning that, without de-randomizing the genetic evolution, it is important to focus on those QEPs with lower associated cost, and avoid spending time optimizing QEPs that are far from the best QEP in the population. HIP is the first technique combining heuristics with genetic query optimizers, and it shows that using simple rules to generate the initial population allows the genetic optimizer to quickly generate good-fitted QEPs, improving the speed and the quality of the optimizer. The combination of both techniques, which are orthogonal, is very simple and it is shown to outperform the best random-walk approach presented in the literature. All in all, we show that, for very large join queries, as the number of relations increases it is advisable to use genetic methods based on beam search strategies, rather than random-walk techniques.

References

1. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt.verlag, Jan. 1998.
2. K. Bennett, M. C. Ferris, and Y. E. Ioannidis. A genetic algorithm for database query optimization. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 400–407, San Mateo, CA, 1991. Morgan Kaufman.
3. S. Chaudhuri and U. Dayal. Data warehousing and OLAP for decision support. *SIGMOD'97: In Proceedings of the ACM SIGMOD international conference on Management of data*, pages 507–508, 1997.
4. Y. E. Ioannidis and Y. Kang. Randomized algorithms for optimizing large join queries. In *SIGMOD '90: Proc. of the 1990 ACM SIGMOD international conference on Management of data*, pages 312–321, New York, NY, USA, 1990. ACM Press.
5. Y. E. Ioannidis and E. Wong. Query optimization by simulated annealing. In *SIGMOD '87: Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, pages 9–22, New York, NY, USA, 1987. ACM Press.

6. A. Kemper, D. Kossmann, and B. Zeller. Performance tuning for sap r/3. *IEEE Data Eng. Bull.*, 22(2):32–39, 1999.
7. R. Krishnamurthy, H. Boral, and C. Zaniolo. Optimization of nonrecursive queries. In *VLDB*, pages 128–137, 1986.
8. V. Muntés-Mulero, J. Aguilar-Saborit, C. Zuzarte, and J.-L. Larriba-Pey. Cgo: a sound genetic optimizer for cyclic query graphs. In *Proc. of ICCS 2006*, pages 156–163, Reading, UK, May 2006. Springer-Verlag.
9. V. Muntés-Mulero, J. Aguilar-Saborit, C. Zuzarte, V. Markl, and J.-L. Larriba-Pey. Genetic evolution in query optimization: a complete analysis of a genetic optimizer. Technical Report UPC-DAC-RR-2005-21, Dept. d'Arqu. de Comp. Universitat Politècnica de Catalunya (<http://www.dama.upc.edu>), 2005.
10. V. Muntés-Mulero, J. Aguilar-Saborit, C. Zuzarte, V. Markl, and J.-L. Larriba-Pey. An inside analysis of a genetic-programming optimizer. In *Proc. of IDEAS '06*, December 2006.
11. V. Muntés-Mulero, M. Pérez-Cassany, J. Aguilar-Saborit, C. Zuzarte, and J.-L. Larriba-Pey. Parameterizing a genetic optimizer. In *Proc. of DEXA '06*, pages 707–717, September 2006.
12. PostgreSQL. <http://www.postgresql.org/>.
13. P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34. ACM Press, 1979.
14. E. J. Shekita, H. C. Young, and K.-L. Tan. Multi-join optimization for symmetric multiprocessors. In R. Agrawal, S. Baker, and D. A. Bell, editors, *19th International Conference on Very Large Data Bases, August 24–27, 1993, Dublin, Ireland, Proceedings*, pages 479–492. Morgan Kaufmann, 1993.
15. M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and randomized optimization for the join ordering problem. *VLDB Journal: Very Large Data Bases*, 6(3):191–208, 1997.
16. M. Stillger and M. Spiliopoulou. Genetic programming in database query optimization. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 388–393, Stanford University, CA, USA, 28–31 July 1996. MIT Press.
17. A. Swami. Optimization of large join queries: combining heuristics and combinatorial techniques. In *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 367–376. ACM Press, 1989.
18. A. Swami and A. Gupta. Optimization of large join queries. In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 8–17, New York, NY, USA, 1988. ACM Press.
19. A. N. Swami and B. R. Iyer. A polynomial time algorithm for optimizing join queries. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 345–354, Washington, DC, USA, 1993. IEEE Computer Society.
20. Y. Tao, Q. Zhu, C. Zuzarte, and W. Lau. Optimizing large star-schema queries with snowflakes via heuristic-based query rewriting. In *CASCON '03: Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*, pages 279–293. IBM Press, 2003.

Trademarks. IBM is a registered trademark of International Business Machines Corporation in the United States, other countries, or both. Intel and Intel Xeon are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Cost-Based Query Optimization for Multi Reachability Joins

Jiefeng Cheng, Jeffrey Xu Yu, and Bolin Ding

The Chinese University of Hong Kong, China
`{jfcheng,yu,blding}@se.cuhk.edu.hk`

Abstract. There is a need to efficiently identify reachabilities between different types of objects over a large data graph. A reachability join (*R*-join) serves as a primitive operator for such a purpose. Given two types, *A* and *D*, *R*-join finds all pairs of *A* and *D* that *D*-typed objects are reachable from some *A*-typed objects. In this paper, we focus on processing multi reachability joins (*R*-joins). In the literature, the up-to-date approach extended the well-known twig-stack join algorithm, to be applicable on directed acyclic graphs (DAGs). The efficiency of such an approach is affected by the density of large DAGs. In this paper, we present algorithms to optimize *R*-joins using a dynamic programming based on the estimated costs associated with *R*-join. Our algorithm is not affected by the density of graphs. We conducted extensive performance studies, and report our findings in our performance studies.

1 Introduction

With the rapid growth of World-Wide-Web, new data archiving and analyzing techniques bring forth a huge volume of data available in public, which is graph structured in nature including hypertext data, semi-structured data and XML [1]. A graph provides great expressive power for people to describe and understand the complex relationships among data objects. As a major standard for representing data on the World-Wide-Web, XML provides facilities for users to view data as graphs with two different links, the parent-child links (document-internal links) and reference links (cross-document links). In addition, XLink (XML Linking Language) [7] and XPointer (XML Pointer Language) [8] provide more facilities for users to manage their complex data as graphs and integrate data effectively. Besides, RDF [3] explicitly describes semantical resource in graphs.

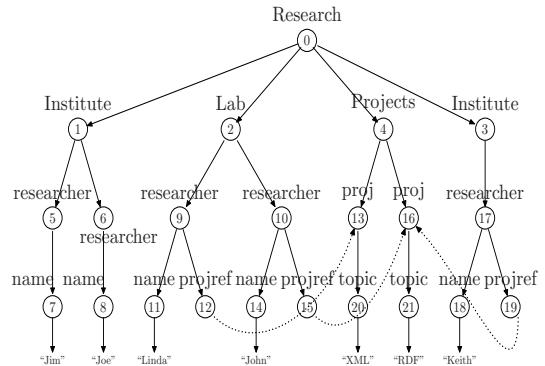
Upon such a graph, a primitive operation, *reachability join* (or simply *R*-join) was studied [1][2]. In brief, a reachability join, $A \rightarrow D$, denoted *R*-join, is to find all the node-pairs, (a, d) , in the underlying large data graph such that *d* is reachable from *a*, denoted $a \sim d$, and the labels of *a* and *d* are *A* and *D* respectively. *R*-joins help users to find information effectively without requesting them to fully understand the schema of the underlying graph. We explain the need of such *R*-join using an XML example. In Figure 1, it shows a graph representation (Figure 1(b)) for an XML data (Figure 1(a)). In Figure 1(b), solid links represent document-internal links whereas dashed links represent cross-document links. We consider Figure 1(b) as a graph with all links being treated in the same way. With *R*-join, we can easily find all the topics that a researcher

```

<Research>
  ...
  <Institute>
    <researcher>
      <name>Jim</name>
    </researcher>
    <researcher>
      <name>Joe</name>
    </researcher>
  </Institute>
  <Lab>
    <researcher>
      <name>Linda</name>
      <projectref idref="proj1" />
    </researcher>
    <researcher>
      <name>John</name>
      <projectref idref="proj2" />
    </researcher>
  </Lab>
  <institute>
    <researcher>
      <name>Keith</name>
      <projectref idref="proj2" />
    </researcher>
  </institute>
  <proj>
    <project id="proj1">
      <topic>XML</topic>
    </person>
    <project id="proj2">
      <topic>RDF</topic>
    </person>
  </proj>
</Research>

```

(a) XML Data



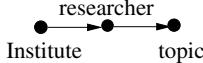
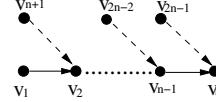
(b) XML Data Graph

Fig. 1. An Example

is interested in using `researcher` \rightarrow `topic`. However, it would be difficult for a user to find the same information using XPath queries, because XPath supports document-internal links using a descendants-or-self-axis operator `//` and cross-document links using value-matching based on a notion called `ID`/`IDREF` in XML. It cannot find such information using an XPath query, `researcher` $//$ `topic`, because `topic` is a child of `proj`, and there is an `ID`/`IDREF` from `researcher` to `proj`. XPath requests users to fully understand the schema and understand that the two different kinds of links are processed in two different ways in XML data. In this paper, we focus ourselves on optimizing and processing multi *R*-join queries.

A query with more than one *R*-joins can be naturally be represented by a query graph. The existing approaches [5][12] extended the well-known tree-specific method, namely, twig-stack join algorithm [4], to process such a query graph over a DAG. We observed that this approach is very sensitive to the density of the underlying DAG. In this paper, we propose a dynamic programming approach that optimizes multi *R*-joins in a similar fashion as to optimize multi joins, based on the estimated costs associated with an *R*-join. The advantage of our approach is that it is not sensitive to the density of the underlying DAG. We conducted extensive experimental studies on multi *R*-join queries using large XMark benchmark dataset [9], which confirms the efficiency of our approach.

The rest of paper is organized as follows. Section 2 gives our problem statement on multi reachability join query processing. Section 3 briefly review the existing technique which extended the well-known twig-stack join algorithm. Together with the motivation of our approach, we discuss drawbacks of such an approach for multi *R*-join queries processing. In Section 4, we review the multiple interval encoding for DAGs, followed by discussions on our cost-based approach that optimizes *R*-joins using a dynamic programming approach for multi *R*-join queries. Section 5 reports the performance evaluation on our proposed method. Section 6 concludes this paper.

Fig. 2. A R -join QueryFig. 3. A Example DAG for *TwigStackD*

2 Multi Reachability Joins

We consider a database as a directed node-labeled data graph $G = (V, E, L, \phi)$. Here, V is a set of elements, E is a set of edges, L is a set of labels, and ϕ is a mapping function which assigns a node a label. Given a label $l \in L$, the extent of l is defined as a set of nodes in G whose label is l , denoted $\text{ext}(l)$. Below, we use $V(G)$ and $E(G)$ to denote the set of nodes and the set of edges of a graph G , respectively. Such a data graph example is shown in Figure ②(b).

A reachability join, $A \hookrightarrow D$, called R -join, is to find all the node-pairs, (a, d) , in the data graph G such that d is reachable from a , denoted $a \rightsquigarrow d$, and $\phi(a) = A$ and $\phi(d) = D$. We also use $D \hookleftarrow A$, instead of $A \hookrightarrow D$, if needed. $A \hookrightarrow D \equiv D \hookleftarrow A$. In this paper, we concentrate on processing conjunctive multi R -join queries in the form of

$$A \hookrightarrow B \wedge B \hookrightarrow C \wedge \dots \wedge X \hookrightarrow Y$$

The following holds for R -joins.

- **Asymmetric:** $A \hookrightarrow B \neq B \hookrightarrow A$.
- **Transitive:** If $A \hookrightarrow B \wedge B \hookrightarrow C$ hold, then $A \hookrightarrow C$.
- **Associative:** $(A \hookrightarrow B) \hookrightarrow C \equiv A \hookrightarrow (B \hookrightarrow C)$ ①

A multi R -join query can be represented as a directed query graph, $G_q(V_q, E_q, L_q, \lambda)$. Here, V_q is a set of nodes. The node-label of a node $v \in V_q$ is represented as $\lambda(v)$. An edge $v \rightarrow u$ represents an R -join $A \hookrightarrow D$, where the labels of v and u are A and D , respectively. A graph representation of a multi R -join query, $A \hookrightarrow B \wedge B \hookrightarrow C \wedge C \hookrightarrow D$, is shown in Figure ②.

We evaluate a query graph $G_q(V_q, E_q, L_q, \lambda_q)$ over a data graph $G(V, E, L, \phi)$. The result of the query graph, G_q , denoted $\mathcal{R}(G_q)$, consists of a set of n -ary tuples. A tuple consists of n nodes in the data graph G , if the query graph G_q has n nodes ($|V(G_q)| = n$), in the form of $t = [v_1, v_2, \dots, v_n]$, where there is a one-to-one mapping between v_i in t and u_i in $V(G_q)$ such that $\phi(v_i) = \lambda(u_i)$. In addition, all nodes in the n -ary tuple r satisfy all the reachability join conditions specified in the query graph G_q .

Table 1. The Graph Encoding of ②

l	v	po_v	lv
Institute	1	5	[1 : 5]
Institute	3	20	[12 : 13][17 : 20]
researcher	5	2	[1 : 2]
researcher	6	4	[3 : 4]
researcher	9	10	[6 : 10]
researcher	10	15	[11 : 15]
researcher	17	19	[12 : 13][17 : 19]
topic	20	7	[7 : 7]
topic	21	12	[12 : 12]

Table 2. The Graph Encoding of ⑤

(a) Tree Interval Encoding		(b) SSPI Index	
v	Interval	v	Interval
1	[2 : 11]	13	[17 : 20]
3	[34 : 41]	16	[27 : 30]
4	[42 : 43]	17	[35 : 40]
5	[3 : 6]	19	[38 : 39]
6	[7 : 10]	20	[18 : 19]
9	[13 : 22]	21	[28 : 29]
10	[23 : 32]		

v	preds
13	{4}
16	{19, 4}
20	{13}
21	{16}

¹ The chain query $A \hookrightarrow B \wedge B \hookrightarrow C \wedge \dots \wedge X \hookrightarrow Y$ is abbreviated to $A \hookrightarrow B \hookrightarrow C \hookrightarrow \dots \hookrightarrow X \hookrightarrow Y$.

Example 1. Fig. 2 represents a simple multi R-join query as a directed graph. This query graph has a node labeled **Institute**, a node labeled **researcher** and a node labeled **topic**. And two edges are in the query graph. The edge from **Institute** node to **researcher** node requires that the data node pair (i, r) , $i \in \text{ext}(\text{Institute})$ and $r \in \text{ext}(\text{researcher})$, such that $i \rightsquigarrow r$, should be returned; in the same time, the edge from **researcher** node to **topic** node requires that the data node pair (r, t) , $r \in \text{ext}(\text{researcher})$ and $t \in \text{ext}(\text{topic})$, such that $r \rightsquigarrow t$, should be returned.

3 Motivation

Recently, as an effort to extend Twig-Join in [4] to be workable on graphs, Chen et al. studied multi R-join query processing(called pattern matching) over a directed acyclic graph (DAG) in [5]. As an approach along the line of Twig-join [4], Chen et al. used the interval-based encoding scheme, which is widely used for processing queries over an XML tree, where a node v is encoded with a pair $[s, e]$ and s and e together specify an interval. Given two nodes u and v in an XML tree, u is an ancestor of v , $u \rightsquigarrow v$, if $u.s < v.s$ and $u.e > v.e$ or simply u 's interval contains v 's.

The test of a reachability relationship in [5] is broken into two parts. First, like the existing interval-based techniques for processing pattern matching over an XML tree, they first check if the reachability relationships can be identified over a spanning tree generated by depth-first traversal of a DAG. Table 2(a) lists the intervals from a spanning tree over the DAG of our running example. Second, for the reachability relationship that may exist over DAG but not in the spanning tree, they index all non-tree edges (named **remaining edges** in [5]), and all nodes being incident with any such non-tree edges in a data structure called SSPI in [5]. Thus, all predecessor/successor relationships that can not be identified by the intervals alone can be found with the help of SSPI. For our running example, Table 2(b) shows SSPI.

As given in [5], for example, the procedure to find the predecessor/successor relationship of $17 \rightsquigarrow 21$ in the DAG of Fig. 1 as follows. First, it checks the containment of tree intervals for 17 and 21, but there is no such a path between them in the tree. Then, because 21 has entries of predecessor in SSPI, it tries to find a reachability relationship between 17 and all 21's predecessors in SSPI by checking the containment of tree interval for 17 and that of each of 21's predecessors in SSPI recursively.

As shown above, in order to identify a reachability relationship between two nodes, say, a and d , *TwigStackD* need to recursively search on SSPI to check if a predecessor of d can be reached by a . This overhead over a DAG can be costly. Consider the DAG of $2n - 1$ nodes in Fig. 2 where the solid lines are edges in the spanning tree generated by a depth-first search, and dashed lines are the remaining edges. Note that in the SSPI, the entry for v_n contains nodes $v_{n+1}, v_{n+2}, \dots, v_{2n-1}$. Thus to determine the reachability relationship from node v_{2n-1} to node v_n , *TwigStackD* needs $n - 1$ times of checking to see if v_{2n-1} can reach any node in the entry. The cost of processing R-joins queries is considerable high.

We conducted tests to confirm our observations. We generate a DAG by collapsing all strongly connected components in a graph that is obtained using XMark data generator dataset with a factor 0.01 (16K nodes). Here both XML tree edge and ID/IDREF links are treated as the edges in the graph. Fig. 4 shows the performance of *TwigStackD*

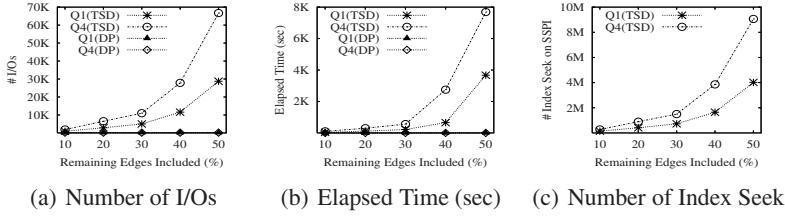


Fig. 4. The Test on DAGs with Increasing Densities

on 5 DAGs, with 10%, 20%, 30%, 40% and 50% of non-tree edges (called remaining edges) as the percentage of the total tree edges in the spanning tree obtained from the graph. The queries used are , Q1 and Q4, which are listed in Fig. 7(a) and (d). In Fig. 4, $Q(TSD)$ and $Q(DP)$ are the processing costs to process Q using Chen *TwigStackD* and our dynamic programming approach, respectively.

Fig. 4(a) shows the I/O number when more remaining edges are added to the underlying DAG. As an example, for query Q4(TSD), the I/O number increased by 4,606 from 10% to 20% on the y-axis, while it increased by 38,881 from 40% to 50% on the y-axis. When 5 times of number of remaining edges is included, the I/O number increases about 35 times. As for the number of index seeks in SSPI, namely the number of times to seek an leaf page from the B+-Tree that implements the SSPI, which is showed in Fig. 4(c), this value increased by 616,052 from 10% to 20% on the y-axis, while it increased by 5,201,991 from 40% to 50% on the y-axis. The correlation coefficient for such two types of measurements is as hight as above 0.999, which speaks that such an behavior for the number of I/Os during processing is mainly caused by the number of index seek of SSPI. Similar situation for processing time can also be observed in Figure 4(c), since the I/O number is the dominating factor for total processing cost. This test empirically showed that *TwigStackD* performs better for DAGs with fewer remaining edges, but its performance degrades rapidly when more edges being included in the underneath DAG.

Fig. 4(a) and (b) also show the efficiency of our dynamic programming approach. Our approach is not so sensitive as *TwigStackD* is to the density of the DAG. For Q4, our approach only uses less than 200 number of I/O access, and 1 second processing time.

4 A New Dynamic Programming Approach

Dynamic programming has been widely used and studied as an effective paradigm for query optimization [10]. In this section, we show how to use dynamic programming to optimize and process multi R -joins queries. In brief, we use an R -join algorithm [11] that uses a multiple interval encoding scheme [2] for processing R -joins over a DAG. Below, first, we discuss the R -join algorithm [11], and how to extend it to process multi R -joins. Then, we will discuss R -join size estimation, and give our optimization approach based on dynamic programming.

4.1 An R -Join Algorithm Based on a Multiple Interval Encoding

Agrawal et al. proposed an interval-based coding for encoding DAG [2]. Unlike the approaches that assign a single code, $[s : e]$, for every node in a tree, Agrawal et al.

assigned a set of intervals and a postorder number for each node in DAG. Let $I_u = \{[s_1 : e_1], [s_2 : e_2], \dots, [s_n : e_n]\}$ be a set of intervals assigned to a node u , there is a path from u to v , $u \rightsquigarrow v$, if the postorder number of v is contained in an interval, $[s_j : e_j]$ in I_u . The interval-based coding for the graph in Figure I(b) is given in Table II. For the same example of $17 \rightsquigarrow 21$ in the DAG of Fig. I, it can be identified by the 21's $poId$, 12, and one interval associated with 17, $[12 : 13]$, since 12 is contained in $[12 : 13]$.

Based on [2], Wang et al. studied processing R -join over a directed graph [III]. In brief, given a directed graph, G . First, it constructs a DAG G' by condensing all strongly connected component in G as a node in G' . Second, it generates encoding for G' based on [2]. All nodes in a strongly connected component in G share the same code assigned to the corresponding representative node condensed in G' . Given an R -join, $A \leftarrow D$, two lists $Alist$ and $Dlist$ are formed respectively. $Alist$ encodes every node v as $(v, s:e)$ where $[s : e] \in I_v$. A node of A has n entries in the $Alist$, if it has n intervals. $Dlist$ encodes each node v as (v, po_v) where po_v is the postorder number. Note: $Alist$ is sorted on the intervals $[s : e]$ by the ascending order of x and then the descending order of y , and $Dlist$ is sorted by the postnumbers in ascending order. Wang et al. proposed to merge-join the nodes in $Alist$ and $Dlist$ and to scan the two lists once.

4.2 Multi R -Joins Processing

It is important to know that some necessary extension is needed to use the R -join algorithm [II] to process multi R -joins. Consider $A \leftarrow D \wedge D \leftarrow E$. For processing $A \leftarrow D$, $Dlist$ needs to be sorted based on the postnumbers, because D is descendant. For processing $D \leftarrow E$, $Dlist$ needs to be sorted based on s followed by e for all $(v, s:e)$, because D is a successor. Also, recall, for $A \rightsquigarrow D$, $Alist$ needs to encode every node v as $(v, s:e)$ where $[s : e] \in I_v$, which means there is a blocking between the two consecutive R -joins, $A \leftarrow D$ followed by $D \leftarrow E$, and we need to generate a new $Alist$ from the output of the previous R -join, $A \leftarrow D$, in order to carry out the next R -join, $D \leftarrow E$. Thus, The intervals and postnumbers of each node must be maintained in multi R -join processing for regeneration of intermediate $Alist$ or $Dlist$ on the fly. A total three operations are needed during such blocking that enables multi R -join query processing.

- $\alpha(A)$: Given a list of node vectors in the form of (v_1, v_2, \dots, v_l) and each v_i is in the extension associated with A , it attaches each interval $[s, e] \in I_{v_i}$ and obtain a number of $(v_1, v_2, \dots, v_l, [s : e])$ from every vector (v_1, v_2, \dots, v_l) and sorts the resulting list to obtain an $Alist$ from these vector. For example, considering execution of two consequent R -joins, $Institute \leftarrow researcher$ and $researcher \leftarrow topic$, to process the query of our running example, the first R -join $Institute \rightarrow researcher$ will produce a set of temporary results A' , $\{(1, 5), (1, 6), (3, 17)\}$. In order to make the proper input for the second R -join $researcher \rightarrow topic$. An $\alpha(A')$ operation is hence applied and we obtain $\{(1, 5, [1 : 2]), (1, 6, [3 : 4]), (3, 17, [12 : 13]), (3, 17, [17 : 19])\}$, which becomes the input $Alist$ for the second R -join.
- $\delta(D)$: Similarly as α , but it attaches the postnumbers for every vector (v_1, v_2, \dots, v_l) and obtains the $(v_1, v_2, \dots, v_l, [po_{v_i}])$, v_i in the extension associated with D , to form a sorted $Dlist$. For example, considering execution of two consequent R -joins, $researcher \rightarrow topic$ and $Institute \leftarrow researcher$, to process the query of our running example, the first R -join $researcher \rightarrow topic$ will produce a set of temporary results

D' , $\{(9, 20), (10, 21), (17, 21)\}$. In order to make the proper input for the second R -join $Institute \leftarrow researcher$. An $\delta(D')$ operation is hence applied and we obtain $\{(9, 20, [10]), (10, 21, [15]), (17, 21, [19])\}$, which becomes the input $Dlist$ for the second R -join.

- $\sigma(A, D)$: Given a list of node vectors in the form of (v_1, v_2, \dots, v_l) and v_i/v_j in a vector is in the extensions associated with A/D , it select out those vectors satisfying $v_i \leftrightarrow v_j$. This is used to processing an R -join $A \leftarrow D$ when both A nodes and D nodes already present in the partial solution. For example, considering the query in Fig. 7(c) and four consequent R -joins, $I \leftarrow C$, $I \leftarrow P$, $C \leftarrow P$ and $L \leftarrow P$ to evaluate that query, when the processing for $I \leftarrow C$, $I \leftarrow P$ and $C \leftarrow P$ has been done, we only further need a $\sigma(L, P)$ to finish the total evaluation.

We develop the cost function involving those operations during processing for multi R -joins after the description for R -join size estimation.

4.3 R-Join Size Estimation

We introduce a simple but effective way to estimate the answer size for a sequence of R -joins. We need two presumption for our estimation: (1) For any pair-wise R -join, say $A \leftarrow D$, every pair of instance (a, d) , where $a \in ext(A)$ and $d \in ext(D)$, is joinable with the same probability. (2) Consider two R -joins, say $A \leftarrow B$ and $B \leftarrow C$, for any three instance (a, b, c) , where $a \in ext(A)$, $b \in ext(B)$, and $c \in ext(C)$, the two events $E_1 = \{a \text{ is joinable with } b\}$ and $E_2 = \{b \text{ is joinable with } c\}$ are independent.

Suppose the answer size for R -joins $(R_1 \leftarrow R_2 \wedge \dots \wedge R_{i-1} \leftarrow R_i)$ is M and the answer size for the pairwise R -join $R_h \leftarrow R_{i+1}$, where $1 \leq h \leq i$, is N , we will show the answer size for $(R_1 \leftarrow R_2 \wedge \dots \wedge R_{i-1} \leftarrow R_i) \wedge (R_h \leftarrow R_{i+1})$ can be estimated as $\frac{M \times N}{|R_h|}$, where $|R_h|$ is the cardinality for the extension of R_h .

Suppose r_j is an instance from $ext(R_j)$, and let $Join(\cdot)$ denote the event that instances are joinable. Then because presumption (2), we have

$$\Pr(Join(r_1, r_2..r_i, r_{i+1})) = \Pr(Join(r_1..r_i) \wedge Join(r_i, r_h)) = \Pr(Join(r_1..r_i)) \cdot \Pr(Join(r_i, r_h)).$$

And because of presumption (1), we have

$$\Pr(Join(r_1..r_i)) \approx \frac{M}{|R_1| \cdot |R_2| \cdot \dots \cdot |R_i|} \quad \Pr(Join(r_i, r_h)) \approx \frac{N}{|R_h| \cdot |R_{i+1}|}.$$

So the estimated answer size of $(R_1 \leftarrow R_2 \wedge \dots \wedge R_{i-1} \leftarrow R_i) \wedge (R_h \leftarrow R_{i+1})$ can be

$$\begin{aligned} EST &= |R_1| |R_2| \dots |R_i| |R_{i+1}| \Pr(Join(r_1, r_2..r_i, r_{i+1})) \\ &= |R_1| \dots |R_{i+1}| \frac{M}{|R_1| |R_2| \dots |R_i|} \frac{N}{|R_h| |R_{i+1}|} = \frac{M \times N}{|R_h|}. \end{aligned}$$

So we will be able to estimate the answer size for all such R -joins by conveniently memorizing all pairwise R -join size and all label's extension cardinalities in the database catalog.

Example 2. For our running example, the first join is $Institute \leftarrow research$, thus $M = 3$. For $Institute \leftarrow research \leftarrow topic$, since $N = 3$ and $|ext(research)| = 5$, so the estimated result set size is $\frac{3 \times 3}{5} = 1.8$. The same result can be calculated if $research \leftarrow topic$ is taken as the first join.

4.4 The Enumeration Space for Multi R-Joins

We use dynamic programming style optimization to enumerate a set of equivalent plans to evaluate a multi R -join query graph G_q against a database graph G . We briefly outline the procedure of searching such plans and its execution to evaluate G_q .

Given a query graph G_q , only left-deep tree plans are searched as a common practice for a reasonable search space. Recall: in G_q , a node represent a label and an edge represents \hookrightarrow . An R -join, $A \hookrightarrow D$, is represented as an edge from A to D . Initially, subgraphs G_2 with two nodes connected by an edge are considered. Here, $V(G_2) = \{v, u\}$ and $E(G_2) = \{(v, u)\}$ or $E(G_2) = \{(u, v)\}$ depending on whether it is for $v \hookrightarrow u$ or $u \hookrightarrow v$. In the next step, it considers to add one more edge. That is, it considers a subgraph G_3 with three edges, such that E_3 includes all the edges in $E(G_2)$ plus one edge which connects at least one incident node in $V(G_2)$. The last step repeats until it includes all the nodes and edges in the original query graph G_q and we can get a sequence of subgraphs (G_2, G_3, \dots, G_m) and a sequence of edges being added (e_2, e_3, \dots, e_m). Regarding a subgraph in the sequence, say, G_i and the edge to be added to the subgraph, which should be e_i or more specifically, (u_i, v_i) , there are 3 cases:

- Only u_i exists in $V(G_i)$, in this case, an α operation is needed and followed by a join for $u_i \hookrightarrow v_i$ and the cost is calculated as

$$C_I = C_\alpha \cdot |\mathcal{R}(G_i)| + C_{\hookrightarrow} (\epsilon \cdot |\mathcal{R}(G_i)| + |Dlist_{v_i}|)$$

- Only v_i exists in $V(G_i)$, in this case, an δ operation maybe needed and followed by a join for $u_i \hookrightarrow v_i$, since the $Dlist$ for v_i maybe obtained by the output from the preceding join. When δ operation is needed, the cost is calculated as

$$C_H = C_\delta \cdot |\mathcal{R}(G_i)| + C_{\hookrightarrow} (|\mathcal{R}(G_i)| + |Alist_{u_i}|)$$

The first term in C_H can be eliminated if no δ operation needed.

- Both u_i and v_i exist in $V(G_i)$, in this case, an σ operation is needed. The cost is calculated as $C_{III} = C_\sigma \cdot |\mathcal{R}(G_i)|$.

In these cost formulae, values for $|Alist_{u_i}|$ and $|Dlist_{v_i}|$ are obtained from the statistics in database catalog. The intermediate result by evaluating the query graph G_i is represented as $\mathcal{R}(G_i)$. We estimate the value of $|\mathcal{R}(G_i)|$ according to section 4.3. The explanation for other factors are as follows,

- C_α : factor to approximate the cost of α operation by the cardinality of the node vectors;
- C_δ : factor to approximate the cost of δ operation by the cardinality of the node vectors;

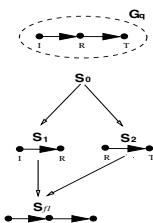


Fig. 5. Searching for an Optimal Plan using DP

Start	End	R-join	Result Size	Cost
s_0	s_1	$I \hookrightarrow R$	3	10
s_0	s_2	$R \hookrightarrow T$	3	10
s_1	s_{f1}	$R \hookrightarrow T$	1	21
s_2	s_{f1}	$I \hookrightarrow R$	1	19

Fig. 6. DP on G_q

- C_σ : factor to approximate the cost of σ operation by the cardinality of the node vectors;
- C_{\rightarrow} : factor to approximate the cost of R -join operation by the sum of two lists' length;
- ϵ : factor to approximate the length of an *Alist* by the cardinality of the node vectors.

4.5 Our Dynamic Programming Algorithm

In our dynamic programming style optimization, two basic components in the solution space are *statuses* and *moves*.

- A status, S , specifies a subquery, G_s , as an intermediate stage in generating a query plan. To be more specific, a subquery of G_q is a subgraph G_s , where $V(G_s) \subseteq V(G_q)$ and $E(G_s) \subseteq E(G_q)$. Note: G_s does not necessarily be a connected graph if without the left-deep tree restriction.
- A move from one status (subquery G_{s_i}) to another status (subquery G_{s_j}) considers an additional edge (R -join) in G_{s_j} that does not appear in G_{s_i} , toward finding the entire query plan for G_q . The next status is determined based on a cost function which results in the minimal cost, in comparison with all possible moves. The process of moving from one status to another results in a left-deep tree which is the R -join order selection result.

We can estimate the cost for each move by those cost formulae in Sec. 4.4. Each status S is associated with a cost function, denoted $cost(S)$, which is the minimal accumulated estimated cost to move from the initial status S_0 to the current status S . Such accumulated cost of a sequence of moves from S_0 to S is the estimated cost for evaluating the subquery G_S being considered under the current status S . Our goal for dynamic programming is to find the sequence of moves from the initial status S_0 toward the final status S_f with the minimum cost, $cost(S_f)$, among all the possible sequences of moves. This method is quite strait forward and its search space is bounded by 2^m .

Our algorithm is outlined in Algorithm II. We simply apply Dijkstra's algorithm for the shortest path problem into our search space, aiming to find a "shortest" path from S_0 to any S_f , where nodes represent statuses, edges represent moves, and the length of an edge is the cost of one move. We omit further explanation about Algorithm II.

Algorithm 1. DP Algorithm to Generate Plan

l is a priority queue of status, sorting statuses in the increasing order of $cost(S)$.

- 1: **Initialize** queue l as \emptyset ;
 - 2: **Add** S_0 into l ;
 - 3: **while** l is NOT empty **do**
 - 4: $S = l.first$;
 - 5: **Delete** $l.first$ from l ;
 - 6: **if** S is a Final Status **then**
 - 7: **Output** plan \mathcal{P} backward from l ; **Terminate** this Algorithm;
 - 8: **for** each move from S to S' **do**
 - 9: **if** $S' \notin l$ **then**
 - 10: **Insert** S' into l ;
 - 11: **else**
 - 12: **Update** $cost(S')$ and l ;
-

Example 3. For our running example, Figure 5 shows two alternative plans for evaluating the query $I \rightarrow R \rightarrow T$, both containing two moves. The status S_0 is associated with a NULL graph, while S_1 and S_2 are respectively associated with two two graphs with two connected nodes, and S_3 is associated with the G_q and thus to be a final status. Details steps in the searching for an optimal plan is showed in Figure 6 where each row of the table lists a move in the solution space. The first column is the status where to start the move and the second column is the status where the move reaches. The third column is the R-join that will be processed in that move, while the number of results generated after the R-join is the fourth column.

5 Performance Evaluation

In this section, we conducted two sets of tests to show the efficiency of our approach. The first set of tests is designed to compare our dynamic programming approach (denoted DP) with algorithm [5] (denoted TSD). The second set of tests further confirms the ability to scale of our approach. We implemented all the algorithms using C++ on top of a Minibase-based variant deployed in Windows XP. We configure the buffer of the database system to be 2MB. A PC with a 3.4GHz processor, 2GB memory, and 120G hard disk running Windows XP is used to carry out all tests.

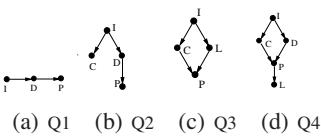


Fig. 7. R-join Query Graphs

Dataset	V	E	I	I / V
20M	307,110	352,214	453,526	1.478
40M	610,140	700,250	901,365	1.477
60M	916,800	1,003,437	1,360,559	1.484
80M	1,225,216	1,337,378	1,816,493	1.483
100M	1,666,315	1,756,509	2,269,465	1.485

Fig. 8. Datasets Statistics

We generated 20M, 40M, 60M, 80M and 100M size XMark datasets [9] using 5 different factors, 0.2, 0.4, 0.6, 0.8, and 1.0 respectively, and named each dataset by its size. In these XML documents, we treat parent-child edges and ID/IDREF edges without difference to obtain graphs and collapse the strong connected components in graphs to get DAGs. The details of the datasets are given in Fig. 8. In Fig. 8 the first column is the dataset name. The second and third columns are the node number and edge number of the resulting DAG respectively. The forth column is the multiple interval labeling size, while the last column shows the average number of intervals per node in the DAG. Throughout all experiments, we use the 4 multi R-join join queries listed in Fig. 7 where the label I stands for *interest*, C for *category*, L for *listitem*, D for *description* and P for *parlist*.

5.1 TwigStackD v.s. DP

We test all queries over the same dataset described in Section 3 for the purpose of compare *TwigStackD* algorithm to our approach. We show two set of figures that show the elapsed time, number of I/Os and memory used to process each query.

² Developed at Univ. of Wisconsin-Madison.

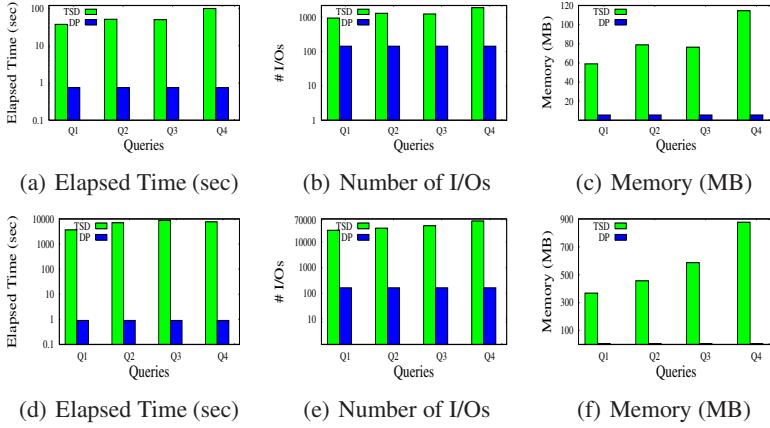


Fig. 9. Compare on the DAG with 10% and 50% Remaining Edge Included

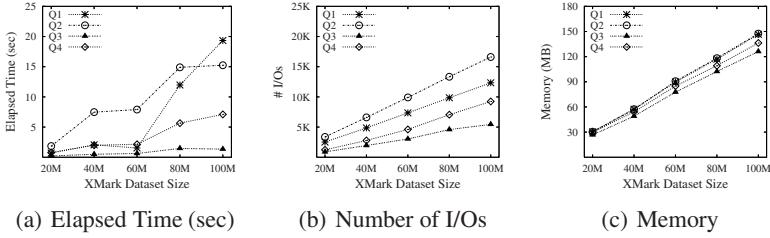
The first set of figures shows the performance on the DAG with 10 percent remaining edges added, which are listed in Fig. 9(a)-(c), and the second set of figures show the performance on the DAG with 50 percent remaining edges added, which are listed in Fig. 9(d)-(f).

As shown in Fig. 9, our approach significantly outperforms *TwigStackD*, in terms of elapsed time, number of I/O accesses, and memory consumption. The sharp difference becomes even greater for a denser DAG, due to the rapid performance degradation of *TwigStackD* when the edge number in the DAG increases. For example, consider Q3, *TwigStackD* used 16.7 times of elapsed time and 8.7 times of I/O accesses than those for our approach when 10 percent remaining edges being added, but when 50 percent remaining edges being added, the two rates become 2922.3 and 266.4 respectively. The memory usage of *TwigStackD* is unstable, and can range from 60MB to 900MB for the 4 queries, because *TwigStackD* needs to buffer every node that can potentially participate in any final solution and thus largely depends on the solution size. And it can also be observed that the larger query needs more memory for the increased needs of buffer pools by *TwigStackD* generally.

5.2 Scalability Test of Our Approach

Because *TwigStackD* does not scale well, in this section, we report the scalability of *DP*. With the size of the dataset increasing from 20M to 100M, we tested the scalability performance for our approach and Fig. 10 shows the results.

Both the number of I/Os and memory usage increase evenly as the size of underlying DAGs increases. However, for the processing time of each query when the data size increased, its variation is not so uniformly. A main reason for this observation is the CPU overhead caused by sorting which is required in α and δ operations, for different distribution of the data may result different join processing order, hence different number of those operations for the same query. However, there is no abrupt change for the processing and the overall performance is still acceptable and all queries can be done within tens of seconds.

Fig. 10. Scalability Test on *DP*

6 Conclusion

In this paper, we studied query processing of multi reachability joins (*R*-joins) over a large DAG. The most up-to-date approach, *TwigStackD* algorithm, uses a single interval encoding scheme. *TwigStackD* assigns to each node in a DAG a single interval based on a spanning tree it obtains from the DAG, and builds a complimentary index called SSPI. It uses a twig-join algorithm to find matches that exist in the spanning tree and buffers all nodes that belong to any solution, in order to find all matches in the DAG, with the help of SSPI. *TwigStackD* has good performance for rather sparse DAGs. But, its performance degrades noticeably when DAG becomes dense, due to the high overhead of accessing edge transitive closures.

We present an approach of using an existing multiple interval encoding scheme that assigns to each node multiple intervals. With the multiple encoding scheme, no additional data structure is needed. We show that optimizing *R*-joins (*R*-join order selection), using dynamic programming with a primitive implementation of *R*-join, can significantly improve the performance, even though such an approach may introduce overhead for feeding the intermediate result of an *R*-join to another. We conducted extensive performance studies and confirmed the efficiency of our *DP* approach. *DP* significantly outperforms *TwigStackD*, and is not sensitive to the density of the underneath DAG.

Acknowledgment

This work was supported by a grant of RGC, Hong Kong SAR, China (No. 418206).

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.
2. R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *Proc. of SIGMOD'89*, 1989.
3. D. Brickley and R. V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. W3C Candidate Recommendation, 2000.
4. N. Bruno and N. K. et. al. Holistic twig joins: optimal xml pattern matching. In *Proc. of SIGMOD'02*.
5. L. Chen and A. G. et. al. Stack-based algorithms for pattern matching on dags. In *Proc. of VLDB'05*.

6. J. Cheng and J. X. Y. et. al. Fast reachability query processing. In *Proc. of DASFAA'06*.
7. S. DeRose, E. Maler, and D. Orchard. XML linking language (XLink) version 1.0. 2001.
8. S. DeRose, E. Maler, and D. Orchard. XML pointer language (XPointer) version 1.0. 2001.
9. A. Schmidt and F. W. et. al. XMark: A benchmark for XML data management. In *Proc. of VLDB'02*.
10. P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proc. SIGMOD'79*, pages 23–34, 1979.
11. H. Wang, W. Wang, X. Lin, and J. Li. Labeling scheme and structural joins for graph-structured XML data. In *Proc. of The 7th Asia Pacific Web Conference*, 2005.
12. H. Wang, W. Wang, X. Lin, and J. Li. Subgraph join: Efficient processing subgraph queries on graph-structured XML document. In *Proc. of WIAM'02*, 2005.

A Path-Based Approach for Efficient Structural Join with Not-Predicates

Hanyu Li, Mong Li Lee, Wynne Hsu, and Ling Li

School of Computing, National University of Singapore
`{leeml,whsu,lil}@comp.nus.edu.sg`

Abstract. There has been much research on XML query processing. However, there has been little work on the evaluation of XML queries involving not-predicates. Such queries are useful and common in many real-life applications. In this paper, we present a model called *XQuery tree* to model queries involving not-predicates and describe a path-based method to evaluate such queries efficiently. A comprehensive set of experiments is carried out to demonstrate the effectiveness and efficiency of the proposed solution.

1 Introduction

Research on XML query processing has been focused on queries involving structural join, e.g., the query “`//dept[/name=”CS”]//professor`” retrieves all the professors in the CS department. However, many real world applications also require complex XML queries containing not-predicates. For example, the query “`//dept[NOT(/name=”CS”)]//professor`” retrieves all the professors who are not from the CS department. We call this class of queries *negation queries*.

A naive method to evaluate negation queries is to decompose it into several normal queries involving structural join operation. Each decomposed query can be evaluated using any existing structural join method [16,7,8,9,12,11], followed by a post processing step to merge the results. This simplistic approach is expensive because it requires repeated data scans and overheads to merge the intermediate results. The work in [10] propose a holistic path join algorithm which is effective for path queries with not-predicates, while [14] develop a method called TwigStackList \neg to handle a limited class of twig queries with not-predicates, i.e., queries with answer nodes above any negative edge.

In this paper, we propose a path-based approach to handle a larger class of negation queries efficiently, i.e., queries with answer nodes both above and below negative edges. We introduce a model called XQuery tree to model queries involving negated containment relationship. We utilize the path-based labeling scheme in [11] for queries involving not-predicates. Experiment results indicate that the path-based approach is more efficient than TwigStackList \neg [14].

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 illustrates the drawback of the TwigStackList \neg method. Section 4 describes the proposed path-based approach. Section 5 gives the experimental results and we conclude in Section 6.

2 Related Work

The structural join has become a core operation in XML queries [46, 78, 9, 12, 11]. The earliest work [2] use a sort-merge or a nested-loop approach to process the structural join. Index-based binary structural join solutions employ B^+ -tree [7], XB-tree [6], XR-tree [8] to process queries efficiently. Subsequent works extend binary structural join to holistic twig join. Bruno et al. [6] propose a holistic twig join algorithm, *TwigStack*, which aims at reducing the size of the intermediate result and is optimal for ancestor-descendent relationship, while [13] design an algorithm called *TwigStackList* to handle parent-child relationships. The work in [11] design a path-based labeling scheme to reduce the number of elements accessed in a structural join operation.

Al-Khalifa et al. [5] examine how the binary structural join method can be employed to evaluate negation in XML queries. Algorithm *PathStack*– [10] utilizes a boolean stack to answer negation queries. The boolean stack contains a boolean variable "satisfy" which indicates whether the associated item satisfies the sub-path rooted at this node. In this way, a negation query does not need to be decomposed, thus improving the query evaluation process.

Algorithm *TwigStackList*– [14] extends the algorithm *TwigStackList* [13] to handle holistic twig negation queries. *TwigStackList*– also avoids decomposing holistic negation queries into several sub-queries without negations. However, *TwigStackList*– can only process a limited class of negation queries and suffer from high computational cost (see Section 3). In contrast, our approach utilizes the path-based labeling scheme in [11] to filter out unnecessary element nodes efficiently and handles a larger class of negation queries.

3 Motivating Example

TwigStackList– [14] defines a query node as an output node if it does not appear below any negative edge, otherwise, it is a non-output node. Consider query T_1 in Fig. 1(b) where $\{B\}$ is an output node and $\{D, E, F\}$ are non-output nodes. Suppose we issue query T_1 over the XML document Doc_1 in Fig. 1(a) whose element nodes have been labeled using the *region encoding* scheme [4]. *TwigStackList*– associates a list L_B and a stack S_B for the output node B . Element B_1 in the XML document is first inserted into the list L_B . Since B_1 satisfies the not-predicate condition in query T_1 , it is also pushed into the stack S_B . Next, element B_2 is inserted into L_B . B_2 is subsequently deleted from L_B since its descendent element D_1 has child nodes E_2 and F_1 , thus satisfying the sub-query rooted at D in T_1 . The final answer for T_1 is B_1 .

There are two main drawbacks in Algorithm *TwigStackList*–. First, the class of negation queries which can be processed is limited to output nodes occurring above any negative edge. Hence, it cannot handle meaningful complex queries such as T_2 in Fig. 1(c) which retrieves all the matching occurrences of elements B and C such that B is not a child of A and B has child nodes C and D while D has a child node E but does not have a descendant node F (we call nodes B and

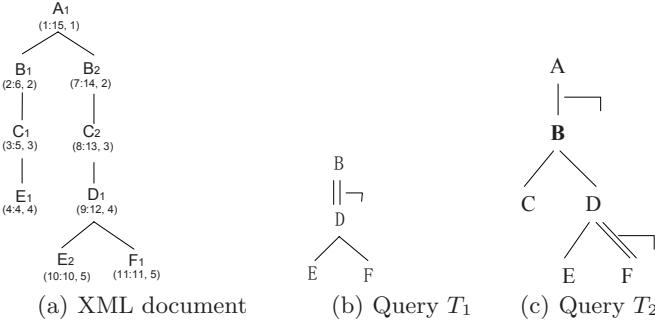


Fig. 1. Example XML document and queries

C projected nodes). Second, TwigStackList \neg may access elements which are not answers to a query. For example, to answer query T_1 , a series of operations is also carried out on element B_2 which is not in the final answer. Our proposed path-based approach aims to overcome these two drawbacks.

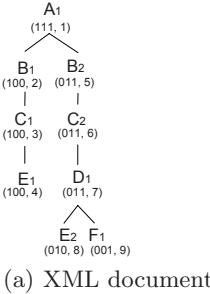
4 Path-Based Approach

The proposed approach to evaluate XML negation queries utilizes the path-based labeling scheme proposed in [11]. We will first review the scheme and introduce the XQuery tree model to represent negation queries. Then we describe the algorithms $PJoin\neg$ and $NJoin\neg$ which removes the unnecessary elements and carries out structural join operation respectively.

4.1 Path-Based Labeling Scheme

The path-based labeling scheme [11] identifies each element node by a pair of (path id, node id). Each text node is identified by a node id. The node id can be assigned using any existing node labeling scheme, e.g, interval-based [12]. A path id is composed of a sequence of bits. We first omit the text nodes from an XML document. Then we find distinct root-to-leaf paths in the XML document by considering only the tag names of the elements on the paths. We use an integer to encode each distinct root-to-leaf path in an XML document. The number of bits in the path id is given by the number of the distinct root-to-leaf element sequences of the tag names that occur in the XML document. Let k denote the number of distinct root-to-leaf paths, hence the path id of an element node has k bits. For a leaf element node, all the bits except for the i th bit, are set to 0, where i is the encoding of the root-to-leaf path on which the leaf node occurs. The path id of a non-leaf element node is given by a bit-or operation on the path ids of all its child element nodes.

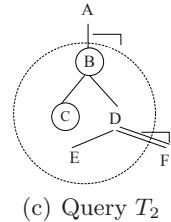
Fig. 2(a) shows the XML document Doc_1 labeled using the path-based labeling scheme. The corresponding encoding table is given in Fig. 2(b).



(a) XML document

Root-to-leaf Path	Encoding
Root/A/B/C/E	1
Root/A/B/C/D/E	2
Root/A/B/C/D/F	3

(b) Encoding Table

(c) Query T_2 **Fig. 2.** Example to illustrate Path Labeling Scheme and XQuery Tree

Let Pid_A and Pid_D be the path ids for elements with tags A and D respectively. If $(Pid_A \& Pid_D) = Pid_D$, then we say Pid_A contains Pid_D . This is called **Path ID Containment**. Li et al. [11] prove that the containment of two nodes can be deduced from the containment of their path ids.

Property I: Let Pid_A and Pid_D be the path ids for elements with tags A and D respectively. If Pid_A contains Pid_D and $Pid_A \neq Pid_D$, then each A with Pid_A must have at least one descendant D with Pid_D .

Consider the element nodes B_2 and E_2 in Doc_1 . The path id 011 for B_2 contains the path id 010 for E_2 since the bit-and operation between 011 and 010 equals to 010 and they are not equal. Therefore, B_2 must be an ancestor of E_2 .

If two sets of nodes have the same path ids, then we need to check their corresponding root-to-leaf paths to determine their structural relationship. For example, the nodes B_1 and E_1 in Doc_1 have the same path id 100. We can decompose the path id 100 into one root-to-leaf path with the encoding 1 since the bit in the corresponding position is 1. By looking up the first path in the encoding table (Fig. 2(b)), we know that B_1 is an ancestor of E_1 .

4.2 XQuery Tree

In this section, we define a model called **XQuery tree** to model queries involving not-predicates. This is accomplished by augmenting the standard XML query pattern tree with two new features: **node projection** and **not** operator.

Definition 1 (XQuery Tree). An XQuery Tree is defined as a tree $T = (V, E)$ where V and E denote the set of nodes and edges respectively.

1. A single edge denotes a parent-child relationship while a double edge denotes an ancestor-descendant relationship.
2. Nodes to be projected are circled.
3. A negated containment relationship between two nodes is specified by putting the symbol “ \neg ” next to the edge. We call such an edge a negated edge.

Fig. 2(c) shows an example negation query modeled using the XQuery tree. The equivalent query specified using the XQuery language is as follows:

```

For $v In //B
Where exists($v/C) and exists($v/D/E) and
      count(A/$v) = 0 and count($v/D//F) = 0
Return {$v} {$v/C}

```

Note that negated edges cannot occur between the projected nodes of a query since they would result in queries that are meaningless, e.g., retrieve all the elements A and B such that A does not contain B . Therefore, we can deduce that given an XQuery tree T , there exists some subtree T' of T such that T' contains all the projected nodes in T and all edges in T' are not negated edges.

Definition 2 (Projected Tree T_P). Let $T = (V, E)$ be an XQuery tree, and S be the set of subtrees $T' = (V', E')$ of T , such that

1. $V' \subseteq V$ and
2. V' contains all the projected nodes in T , and
3. for any $e \in E'$, e is not a negated edge.

The largest T' in S is defined as the projected tree T_P of T .

The projected tree of the XQuery tree in Fig. 2(c) is shown within the dashed circle. Given an XQuery tree T , we define the subtree above T_P as tree T_P^a and the subtree below T_P as T_P^b respectively.

Definition 3 (Tree T_P^a). Given an XQuery tree T , let R be the root node of T_P , and e be the incoming edge of R . We define T_P^a as the subtree obtained from $T - T_R - e$, where T_R denotes the subtree rooted at R .

Definition 4 (Tree T_P^b). Given an XQuery tree T , we define T_P^b as the subtree rooted at C , where C denotes a child node of the leaf nodes of T_P .

In Fig. 2(c), the nodes A and F form the trees T_P^a and T_P^b of T respectively. Note that an XQuery tree T has at most one T_P^a and possibly multiple T_P^b . A tree T_P^a or T_P^b may contain negated edges. However, queries with negated edges in T_P^a or T_P^b may have multiple interpretations. For example, the query “ A does not contain B , and B does not contain C , where C is the projected node” has different semantics depending on the applications. Here, we focus on queries whose subtrees T_P^a and T_P^b do not contain any negated edges.

4.3 Algorithm $PJoin^-$

Algorithm $PJoin^-$ [1] filters out unnecessary path ids for queries involving structural join. The main operation in $PJoin$ is the binary path join. A binary path join takes as input two lists of path ids, one for the parent node and the other for the child node. A nested loop is used to find the matching pairs of path ids

based on the path id containment property. Any path id that does not satisfy the path id containment relationship is removed from the lists of path ids of both the parent node and the child node. However, this algorithm does not work well for queries involving not-predicates.

Consider query T_3 in Fig. 3(a) where the lists of path ids have been associated with the corresponding nodes. We assume that the path ids with the same subscripts satisfy the path id containment relationship, i.e., b_2 contains c_2 , etc.

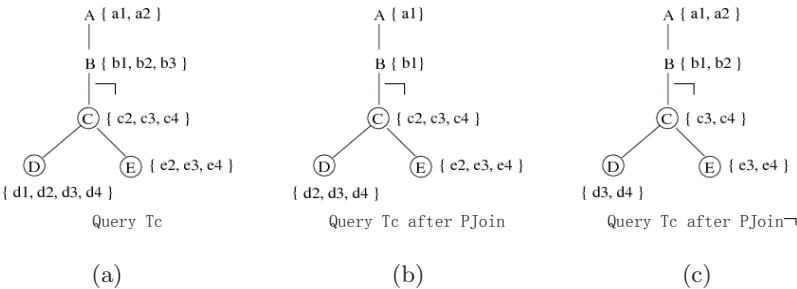


Fig. 3. Example to illustrate Algorithm $PJoin\neg$

Algorithm $PJoin$ will first perform a bottom-up binary path join. The path id lists for nodes C and D are joined. Since the path id d_2 , d_3 and d_4 are contained in the path id c_2 , c_3 and c_4 respectively, d_1 is removed from the set of path ids of D . The path id list of node C is joined with the path id list of node E . No path id is removed since each path id of E is contained in some path id of C . We join the path id list of node B with that of node C . The path ids c_2 and c_3 are contained in the path id b_2 and b_3 respectively. Since there is a not-predicate condition between nodes B and C , the path id b_2 and b_3 need to be removed from the set of path ids of B . Finally, a binary path join between nodes A and B is carried out and the path id a_2 is removed.

Next, Algorithm $PJoin$ carries out a top-down binary path join on T_3 starting from the root node A . The final result is shown in Fig. 3(b). The optimal sets of path ids for the nodes in T_3 is shown in Fig. 3(c). The difference in the two sets of path ids shown in Fig. 3(b) and Fig. 3(c) is because Algorithm $PJoin$ does not apply the constraint that is imposed on nodes A and B to the entire query.

The above example illustrates that the proper way to evaluate a negated containment relationship between path ids is to only update the path ids of the nodes in the projected tree. This leads to the design of Algorithm $PJoin\neg$.

The basic idea behind $PJoin\neg$ (Algorithm II) is that given a query T , we first apply $PJoin$ on T_P^a and T_P^b . The path ids of the leaf node of T_P^a and the root node(s) of T_P^b are used to filter out the path ids of the corresponding nodes in T_P . The input to Algorithm $PJoin\neg$ is an XQuery tree T with a set of projected nodes. We first determine the projected tree T_P of T . Then the $PJoin$ algorithm is carried out on T_P^b and T_P^a (if any) respectively (lines 4-5). Next, a bottom-up binary path join and a top-down binary path join are performed on T_P .

Algorithm 1. $PJoin^{-}$

-
- 1: **Input:** T - An XQuery-tree
 - 2: **Output:** Path ids for the nodes in T
 - 3: Associate every node in T with its path ids;
 - 4: Perform a bottom-up binary path join on T_P^b and T_P^a ;
 - 5: Perform a top-down binary path join on T_P^b and T_P^a ;
 - 6: Perform a path anti-join between the root node(s) of T_P^b and their parent node(s) if necessary;
 - 7: Perform a bottom-up binary path join on T_P ;
 - 8: Perform a path anti-join between the leaf node of T_P^a with its child node if necessary;
 - 9: Perform a top-down binary path join on T_P ;
-

(lines 7, 9). Each binary path join operation is followed by a path antijoin operation (lines 6, 8). A path antijoin takes as input two lists of path ids, but one list of path ids is for reference; only path ids in the other list need to be removed if necessary. In line 6(8), the Algorithm $PJoin^{-}$ utilizes the root(leaf) nodes of T_P^b (T_P^a) to filter out the path ids of their parent(child) node(s).

Note that if the set of path ids for the root node (leaf node) of T_P^b (T_P^a) contains some path id whose corresponding element node is not a result of T (super *Pid* set), then the path antijoin operation in Lines 6 (8) of Algorithm **I** is skipped. This is because the super *Pid* set of the root node (leaf node) of T_P^b (T_P^a) could erroneously remove path ids from its parent node (child node), and we may miss some correct answers in the final query result.

Consider again query T_3 in Fig. 3(a). The projected tree is the subtree rooted at node C . A $PJoin$ is first performed on tree T_P^a which contains nodes A and B . The set of path ids for B obtained is $\{b_1, b_2\}$. Next, bottom-up path join is carried out on T_P . Since T_P^a is a simple path query without value predicates, the path id set associated with B is not a super *Pid* set according to the discussion in **II**. Then we can perform a path anti-join between nodes B and C . This step eliminates c_2 from the path id set of C since c_2 is contained in b_2 . Finally, a top-down path join is performed on T_P , which eliminates d_1 and d_2 from the set of path ids for D , and e_2 from the set of path ids for E . The final result after $PJoin^{-}$ is shown in Fig. 3(c).

4.4 Algorithm $NJoin^{-}$

We retrieve the elements with path ids output by Algorithm $PJoin^{-}$ and apply Algorithm $NJoin^{-}$ on these elements to obtain the result of the negation queries.

Algorithm **2** shows the details of $NJoin^{-}$ method. If the negation query with T_P^a is null, Algorithm $NJoin^{-}$ will use the method $TwigStackList^{-}$ **[4]** to calculate the final result. This is because $TwigStackList^{-}$ cannot handle queries with T_P^a as illustrated in Section **3**. Otherwise, we will use the holistic structural join in **6** to evaluate the trees T_P^a , T_P^b and T_P , and then merge the results.

Algorithm 2. $NJoin_{\neg}$

```

1: Input: T - An XQuery-tree
2: Output: All occurrences of nodes in  $T_P$ 
3: if  $T_P^a$  is null then
4:   Perform TwigStackList $\neg$  on T;
5: else
6:   Perform holistic structural join on  $T_P^a$ ,  $T_P^b$  and  $T_P$ ;
7:   Merge the intermediate result;
8: end if

```

4.5 Optimality of Path-Based Approach

The optimality of the proposed solution is due to Algorithm $PJoin_{\neg}$. This step can greatly reduce the number of elements accessed by Algorithm $NJoin_{\neg}$.

Consider the query T_1 (Fig. 1(b)) issued over the XML document Doc_1 in Fig. 2(a). The path ids of each node is shown in Fig. 4(a). When Algorithm $PJoin_{\neg}$ is applied on T_1 , the path id $\{100\}$ is removed from the path id set of node E and $\{011\}$ is removed from the path id set of node B (see Fig. 4(b)). There is only one path id left for node B after $PJoin_{\neg}$, which corresponds to element B_1 . Further processing of element B_2 is not needed. Experimental results in the next section indicate that the relatively inexpensive $PJoin_{\neg}$ can greatly filter out the irrelevant element nodes.

The other advantage of our approach is that by decomposing negation queries into three parts (T_P , T_P^a and T_P^b), we can handle an additional class of queries compared to the method TwigStackList \neg .

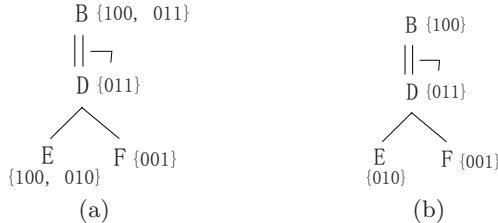


Fig. 4. Example to illustrate optimality of path-based approach

5 Experiment Evaluation

In this section, we examine the performance of the proposed path-based solution for negation queries. We also compare our method with TwigStackList \neg [14]. Both approaches are implemented in C++. All experiments are carried out on a Pentium IV 2.4 GHz CPU with 1 GB RAM. The operating system is Linux 2.4. The page size is set to be 4KB.

We use three real world datasets for our experiments. They are Shakespeare's Plays (SSPlays) [1], DBLP [2] and XMark benchmark [3]. Table 1 shows the characteristics of the datasets and Table 2 gives the query workload.

Table 1. Characteristics of Datasets

Datasets	Size	#(Distinct Elements)	#(Elements)
SSPlays	7.5 MB	21	179,690
DBLP	60.7 MB	32	1,534,453
XMark	61.4 MB	74	959,495

Table 2. Query Workload

	Query	Dataset	Nodes in Result
Q1	//PLAY[NOT(/PROLOGUE)]/EPILOGUE//TITLE	SSPlays	13
Q2	//dblp/article[NOT(/url)]	DBLP	14
Q3	//person[NOT(/creditcard)]	XMark	7618
Q4	//people/person[NOT(/age)]/profile/education	XMark	9568

5.1 Effectiveness of $PJoin^{-\neg}$

We first evaluate the effectiveness of $PJoin^{-\neg}$ in filtering out irrelevant elements for the subsequent $NJoin^{-\neg}$ operation. The following metrics are used:

$$\text{Filtering Efficiency} = \frac{\sum |N_i^p|}{\sum |N_i|}$$

$$\text{Selectivity Rate} = \frac{\sum |N_i^n|}{\sum |N_i|}$$

where $|N_i^p|$ denotes the number of instances for node N_i after $PJoin^{-\neg}$ operation, $|N_i^n|$ denotes the number of instances for node N_i in the result set after $NJoin^{-\neg}$ operation and $|N_i|$ denotes the total number of instances for node N_i in the projected tree of the query.

Fig. 5(a) shows the *Filtering Efficiency* with *Selectivity Rate* for queries Q1 to Q4. The closer the two values are, the more effective $PJoin^{-\neg}$ is for the query. We observe that Algorithm $PJoin^{-\neg}$ is able to remove all the unnecessary elements for queries Q1, Q2 and Q3 and the subsequent $NJoin^{-\neg}$ will not access any element that does not contribute to the final result, leading to optimal query evaluation. Query Q4 has a higher *Filtering efficiency* value than *Query Selectivity* because the query node **person** which is the root node of the subtree rooted at node **age** is a branch node. The set of path ids for **person** is a super *Pid* set. Nevertheless, Algorithm $PJoin^{-\neg}$ remains effective in eliminating unnecessary path ids even for such queries.

Fig. 5(b) and (c) show that the I/O cost and elapsed time of Algorithm $PJoin^{-\neg}$ are marginal compared with $NJoin^{-\neg}$ for queries Q1 to Q4. This is

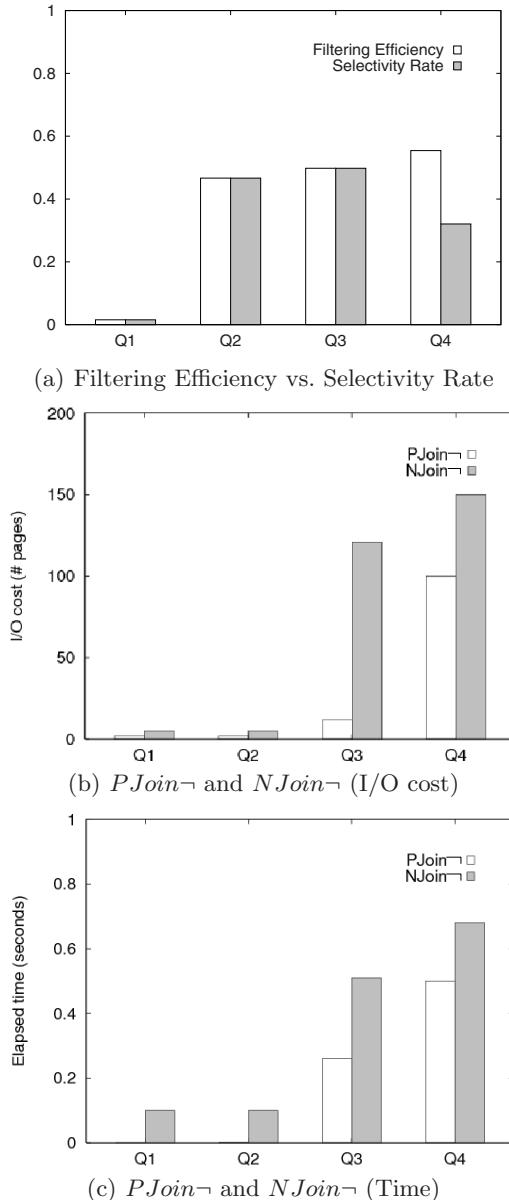


Fig. 5. Effectiveness of $PJoin\neg$

because the sizes of the path lists are much smaller than that of node lists. The time cost of $PJoin\neg$ for queries Q3 and Q4 is slightly more compared to Q1 and Q2 due to a larger number of distinct paths, as well as longer path ids for the XMark dataset.

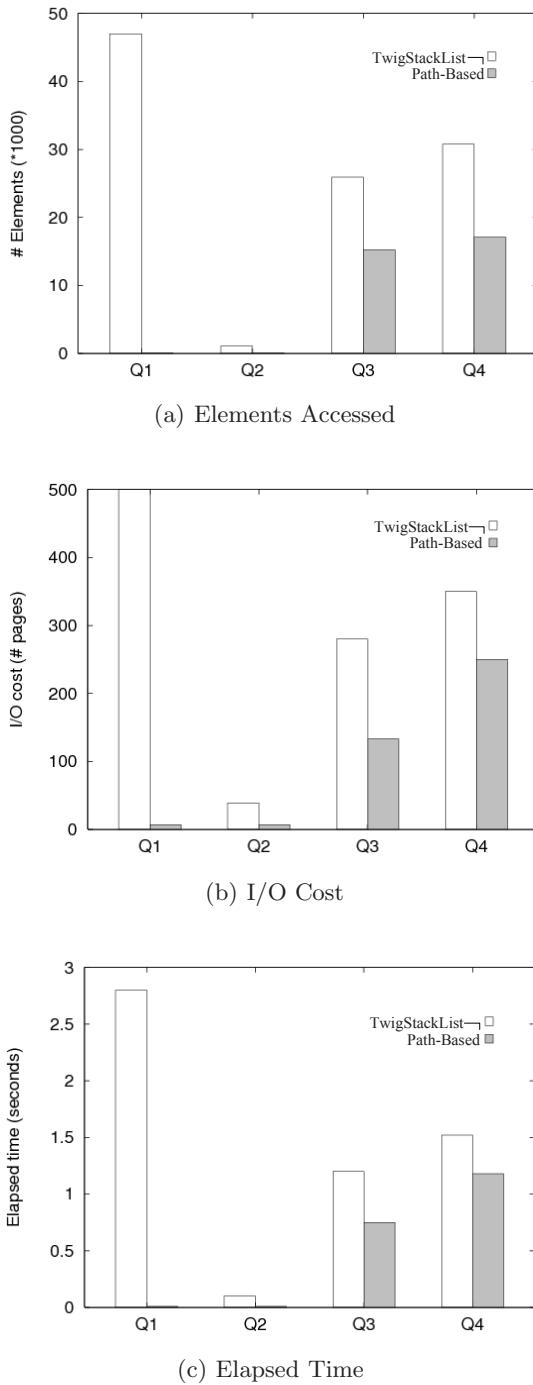


Fig. 6. TwigStackList \neg vs. Path-Based

5.2 Comparative Experiments

In this set of experiments, we compare our solution with TwigStackList \neg [14]. Fig. 6 shows the results. The path-based solution outperforms TwigStackList \neg because Algorithm $PJoin\neg$ is able to greatly reduce the actual number of elements retrieved while TwigStackList \neg is designed to reduce the intermediate result sizes and may access all the elements involved in the queries. For example, TwigStackList \neg must read in the full sets of elements when evaluating Q1.

6 Conclusion

In this paper, we have described a path-based approach to evaluate negation queries. We introduced a model called XQuery tree to model queries involving negated containment relationship. The proposed approach utilizes a path-based labeling scheme to filter out irrelevant elements. Experimental results indicate that the path-based approach is more efficient than TwigStackList \neg and is effective for a larger class of negation queries.

References

1. <http://www.ibiblio.org/xml/examples/shakespeare>.
2. <http://www.informatik.uni-trier.de/~ley/db/>.
3. <http://monetdb.cwi.nl/>.
4. A. Al-Khalifa, H. V. Jagadish, and J. M. Patel et al. Structural joins: A primitive for efficient xml query pattern matching. *IEEE ICDE*, 2002.
5. S. Al-Khalifa and H. V. Jagadish. Multi-level operator combination in xml query processing. *ACM CIKM*, 2002.
6. N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: Optimal xml pattern matching. *ACM SIGMOD*, 2002.
7. S-Y. Chien, Z. Vagena, and D. Zhang et al. Efficient structural joins on indexed xml documents. *VLDB*, 2002.
8. H. Jiang, H. Lu, W. Wang, and B. C. Ooi. Xr-tree: Indexing xml data for efficient structural joins. *IEEE ICDE*, 2003.
9. H. Jiang, W. Wang, and H. Lu. Holistic twig joins on indexed xml documents. *VLDB*, 2003.
10. E. Jiao, T-W. Ling, C-Y. Chan, and P. S. Yu. Pathstack \neg : A holistic path join algorithm for path query with not-predicates on xml data. *DASFAA*, 2005.
11. H. Li, M-L. Lee, and W. Hsu. A path-based labeling scheme for efficient structural join. *International Symposium on XML Databases*, 2005.
12. Q. Li and B. Moon. Indexing and querying xml data for regular path expressions. *VLDB*, 2001.
13. J. Lu, T. Chen, and T-W. Ling. Efficient processing of xml twig patterns with parent child edges: A look-ahead approach. *CIKM*, 2004.
14. T. Yu, T-W. Ling, and J. Lu. Twigstacklist \neg : A holistic twig join algorithm for twig query with not-predicates on xml data. *DASFAA*, 2006.

RRPJ: Result-Rate Based Progressive Relational Join

Wee Hyong Tok, Stéphane Bressan, and Mong-Li Lee

School of Computing
National University of Singapore
`{tokwh, steph, leeml}@comp.nus.edu.sg`

Abstract. Progressive join algorithms are join algorithms that produce results incrementally as input data is available. Because they are non-blocking, they are particularly suitable for online processing of data streams. Reference algorithms of this family are the symmetric hash join, the X-join and more recently, the rate-based progressive join (RPJ).

While the symmetric hash join introduces the idea of a symmetric processing of the input streams but assumes sufficient main memory, the X-Join suggests that the processing can scale to very large amounts of data if main memory is regularly flushed to disk, and a reactive/cleanup phase is triggered for disk-resident data. The X-join flushing strategy is based on a simple largest-first strategy, where the largest partition is flushed to disk. The recently proposed RPJ predicts the main memory tuples or partitions that should be flushed to disk in order to maximize throughput by computing their probabilities to contribute to a result.

In this paper, we discuss the limitations of RPJ and propose a novel extension, called *Result Rate-based Progressive Join* (RRPJ), which addresses these limitations. Instead of computing the probabilities from statistics over the input data, RRPJ directly observes the output (result) statistics. This not only yields a better performance, but also simplifies the generalization of the algorithm to non-relational data such as multi-dimensional data and hierarchical data. We empirically show that RRPJ is effective and efficient and outperforms the state-of-art RPJ. We also investigate the relevance and performance of an adaptive version of these algorithms using amortization parameters.

Keywords: Query Processing, Join Algorithms, Data Streams.

1 Introduction

The universe of network-accessible information is expanding. It is now common practice for applications to process streams of data incoming from remote sources (repositories continuously publishing or sensor networks producing continuous data). An essential operation is the equijoin of two data streams of relational data. Designing an algorithm for such an algorithm must meet a key requirement: the algorithm must be non-blocking (or progressive), i.e. it must be able to produce results as soon as possible, at the least possible expense for the overall throughput.

Several non-blocking algorithms for various operators in general and for the relational equijoin in particular have been proposed [1][2][3][4][5]. These algorithms can be categorized as *heuristic* or *probabilistic* methods. Heuristic methods rely on pre-defined policies for the efficient usage of the available memory; whereas probabilistic methods [6][7] attempt to model the incoming data distribution (values and arrival parameters) and use it to predict the tuples or partitions that are kept in memory in order to produce the maximum number of result tuples. The main thrust in all these techniques lies in the simple idea of keeping useful tuples or partitions (i.e. tuples or partitions likely to produce more results) in memory. Amongst the many progressive join algorithms introduced, one of the state-of-art hash-based progressive join algorithm is the Rate-based Progressive Join (RPJ) [6]. One of the limitations of RPJ is that it is not able to perform well if the data within the partitions are non-uniform, and that it is not straightforward to generalize it for non-relational data. In this paper, we propose the Result-Rate based Progressive join (RRPJ) which overcomes these limitations.

The rest of the paper is organized as follows: In Section 2, we discuss related work and focus on two recent progressive join algorithms, and their strengths and limitations. In Section 3, we present a novel method, called Result Rate-based Progressive Join (RRPJ), which uses a model of the result distribution to determine which tuples to be flushed. We conduct an extensive performance study in Section 4. We conclude in Section 5.

2 Progressive Join Algorithms

In the literature, many equijoin algorithms [2][3][4][5][8][9][10][11] have been proposed. Most of these algorithms considered local datasets, and do not generalize easily to handle unpredictable data arrival common in data streams environment. Many of these equijoin algorithms are based on the seminal work on symmetric hash join's (SHJ) [12]. SHJ assumes the use of in-memory hash tables; an insert-probe paradigm is used to deliver results progressively to users. In the literature, many subsequently proposed progressive relational join algorithms are based on an extended SHJ model, where both in-memory and disk-resident hash partitions are used to store tuples. Whenever memory becomes full, some in-memory tuples need to be flushed to disk to make space for new-arriving tuples. XJoin [3] uses a simple heuristics that flushes the largest partitions. Throughput can be improved if the sacrificed tuples are those with the smallest probability of joining with future tuples, i.e. of contributing to the production of results. This heuristics is the basis of two recent proposals [6][7] that attempt to extrapolate stochastic models of the data and their productivity from the observation of incoming tuples.

2.1 Problem Definition

We consider the problem of performing a relational equijoin between two relational datasets, which are transmitted from remote data sources through an

unpredictable network. Let the two sets of relational data objects be denoted by $R = \{r_1, r_2, \dots, r_n\}$, and $S = \{s_1, s_2, \dots, s_m\}$, where r_i and s_j denotes the i -th and j -th data object from the remote data source respectively. When performing a relational equijoin, with join attribute A , a result is returned when $r_i.A$ is equal to $s_j.A$. Formally, (r_i, s_j) is reported as the result if $r_i.A$ is equal to $s_j.A$. The goal is to deliver initial results quickly and ensure a high result-throughput.

2.2 Rate-Based Progressive Join (RPJ)

RPJ [6] is a hash-based join. It builds a stochastic model based on the tuples' arrival pattern. Whenever memory becomes full, the model is used to determine probabilistically which tuples are least likely to produce tuples with the other incoming data, and hence flushed from memory to disk.

In order to compute the conditional probability that an incoming tuple t belongs to the j -partition, given that t belongs to relation R_i , RPJ keeps track of the total number of tuples from relation i that have arrived and falls into partition j , denoted by $n_i^{total}[j]$. By dividing $n_i^{total}[j]$ over the total number of tuples that have arrived in the system so far, the conditional probability $P(j|R_i)$ can be derived as $P(j|R_i) = \frac{n_i^{total}[j]}{\sum_{j=1}^{n_{part}} n_i^{total}[j]}$. To reduce the need to track conditional probabilities for each values in the domain of the join attribute, RPJ assumes that the data in each partition is uniformly distributed. (i.e. *local uniformity assumption*). The probability $P(R_1)$ and $P(R_2)$ are estimated by maintaining counters n_i^{rcnt} for each relation R_i (initially set to the number of arriving R_i tuples between the initial time interval $[0,1]$). Subsequently, RPJ counts the number of tuples, denoted by $\alpha_i(t)$, between the interval $[t, t+1]$. To more accurately reflect current arrivals, and to reduce the impact from historical arrivals, RPJ updates the value of n_i^{rcnt} to $\lambda \cdot n_i^{rcnt} + (1-\lambda) \cdot \alpha_i(t)$, where λ is a user-tunable parameter (varies between $[0,1]$).

Thus, the arrival probability $p_i^{arr}(v)$ of a tuple belonging to relation R_i and has the value v is then computed as $P_i^{arr}[j] = \frac{n_i^{total}[j]}{\sum_{j=1}^{n_{part}} n_i^{total}[j]} \cdot \frac{n_i^{rcnt}}{n_1^{rcnt} + n_2^{rcnt}}$ (Refer to [6] for the complete proof).

2.3 Locality-Aware (LA) Model

[7] observes that a data stream exhibits reference locality when tuples with specific attribute values have a higher probability of re-appearing in a future time interval. Leveraging this observation, a Locality-Aware (LA) model was proposed, where the reference locality caused by both long-term popularity and short-term correlations are captured. This is described by the following model:

$x_n = x_{n-i}$ (with probability a_i); $x_n = y$ (with probability b , where $1 \leq i \leq h$ and $b + \sum_{i=1}^h a_i = 1$). y denotes a random variable that is independent and identically distributed (IID) with respect to the probability distribution of the popularity, P .

Using this model, the probability that a tuple t will appear at the n -th position of the stream is given by $\text{Prob}(x_n = t | x_{n-1}, \dots, x_{n-h}) = bP(t) + \sum_{j=1}^h a_j \delta(x_{n-j}, t)$

($\delta(x_k, c) = 1$ if $x_k = c$, and it is 0 otherwise). Using the LA model, the marginal utility of a tuple is then derived, and is then used as the basis for determining the tuples to be flushed to disk whenever memory is full.

2.4 Limitations of RPJ and LA Model

In this section, we discuss the limitations of RPJ and the LA model. RPJ rely on the availability of an analytical model deriving the output probabilities from statistics on the input data. This is possible in the case of relational equijoins but embeds some uniformity assumptions that are not necessarily true. It is not able to efficiently handle scenarios in which the data within each partition is non-uniform, which breaks the local uniformity assumption. Consider the two partitions, belonging to dataset R and S respectively, presented in Figure 1. The grayed area is used to denote ranges of data. Suppose in both Figure (a) and (b), N tuples have arrived. In Figure 1(a), the N tuples is uniformly distributed across the entire partitions of each dataset; whereas in Figure 1(b), the N tuples is distributed within a specific numeric range (i.e. areas marked grey). Assume the same number of tuples have arrived for both cases, then $P(1|R)$ and $P(1|S)$ would be the same. However, it is important to note that if partition 1 is selected to be the partition to be kept in memory, the partitions in Figure 1(a) would produce results as predicted by RPJ; whereas the partitions in Figure 1(b) would fail to produce any results. Though RPJ attempts to amortize the effect of historical arrivals of each relation, it assumes that the data distribution remains stable throughout the lifetime of the join, which makes it less useful when the data distribution are changing (which is common in long-running data streams).

The LA model is applied to deal with the approximate sliding window join on relational data. Based on the LA model given in the earlier section, we can see that it relies on determining whether a similar tuple appears in a future position in the data stream. For relational data, a similar tuple could be one that has the same value as a previous tuple. However, for non-relational data, such as spatial data, the notion of similarity between two tuples is more complex, and hence it is not straightforward to extend the LA model to deal with non-relational data types.



Fig. 1. Data in a Partition

3 Result-Rated Based Progressive Join (RRPJ)

In this section, we present a novel method of maintaining statistics over the result distribution, instead of the data distribution. This is motivated by the fact that in most progressive join scenarios, we are concerned with delivering initial results quickly and maintaining a high overall throughput. Hence, the criteria used to determine which tuples need to be flushed to disk whenever memory becomes full should be ‘result-motivated’. In addition, the number of results produced by a partition is reflective of the data distribution of the partitions.

3.1 RRPJ

We propose a novel join algorithm, call Result-Rate Based Progressive Join (RRPJ) (Algorithm 1), which uses information on the result throughput of the partitions to determine the tuples or partitions that are likely to produce results. In Algorithm 1, an arriving tuple is first used to probe the hash partitions of the corresponding data stream in order to produce result tuples. Next, it will check whether memory is full (line 2). If memory is full, it will first compute the Th_i values (i.e value computed by Equation 3) for all the partitions. Partitions with the lowest Th_i values will then be flushed to disk, and the newly arrived tuple inserted. The main difference between the *RRPJ* flushing and *RPJ* is that the Th_i values are reflective of the output (i.e. results) distribution over the data partitions; whereas the RPJ values are based on input the data distribution.

To compute the Th_i values (computed using Equation 3), we track the total number of tuples, n_i (for each partition), that contribute to a join result from the probes against the partition. Intuitively, RRPJ tracks the *join throughput* of each partition. Whenever memory becomes full, we flush n_{flush} (user-defined parameter) tuples from the partition that have the smallest Th_i values, since these partitions have produced the least result so far. If the number of tuples in the partition is less than n_{flush} , we move on to the partition with the next lowest Th_i values.

Given two timestamps t_1 and t_2 ($t_2 > t_1$)and the number of join results produced at t_1 and t_2 are n_1 and n_2 respectively. A straightforward definition of the throughput of a partition i , denoted by Th_i , is given in Equation 1

$$Th_i = \frac{n_2 - n_1}{t_2 - t_1} \text{ (version 1)} \quad (1)$$

From Equation 1, we can observe that since $(t_2 - t_1)$ is the same for all partitions, it suffice to maintain counters on just the number of results produced (i.e. n_1 and n_2). A partition with a high Th_i value will be the partition which have higher potential of producing the most results. However, it is important to note that Equation 1 do not take into consideration the size of the partitions and its impact on the number of results produced. Intuitively, a large partition will produce more results. It is important to note that this might not always be true. For example, a partition might contain few tuples, but produces a lot of results. This partition should be favored over a relatively larger partition which is also

Algorithm 1: RRPJ Join Algorithm

```

Data :  $t$  - Newly Arrived tuple
Result : Result Tuples
1 Use  $t$  to probe hash partitions from other data stream
2 If ( Memory is full() ) {
3   ComputeThValue() ;
4   FlushDataToDisk() }
5 Insert  $t$  into hash table  $HT$ ;

```

producing the same number of results. Besides considering the result distribution amongst the partitions, we must also consider the following: (1) Total number of tuples that have arrived, (2) Number of tuples in each partition, (3) Number of result tuples produced by each partition and (4) Total results produced by the system. Therefore, we use an improved definition for Th_i , given below.

Suppose there are P partitions maintained for the relation. Let N_i denote the number of tuples in partition i ($1 \leq i \leq P$), and R_i denote the number of result tuples produced by partition i . Then, the Th_i value for a partition i can be computed. In Equation 2, we consider the ratio of the results produced to the total number of results produced so far (i.e. numerator), and also the ratio of the number of tuples in a partition to the total number of tuples that have arrived (i.e. denominator).

$$Th_i = \left(\frac{R_i}{\sum_{j=1}^P R_j} \right) / \left(\frac{N_i}{\sum_{j=1}^P N_j} \right) = \frac{R_i \times (\sum_{j=1}^P N_j)}{\left(\sum_{j=1}^P R_j \right) \times N_i} \quad (\text{version 2}) \quad (2)$$

Since the total number of results produced and the total number of tuples is the same for all partitions, Equation 2 can be simplified. This is given in Equation 3.

$$Th_i = \frac{R_i}{N_i} \quad (\text{version 2 - after simplification}) \quad (3)$$

Equation 3 computes the Th_i value w.r.t to the size of the partition. For example, let us consider two cases. In case (1), suppose $N_i = 1$ (i.e. one tuple in the partition) and $R_i = 100$. In case (2), suppose $N_i = 10$ and $R_i = 1000$. Then, the Th_i values for case (1) and (2) are the same. This prevents large partitions from unfairly dominating the smaller partitions (due to the potential large number of results produced by larger partitions) when a choice needs to be made on which partitions should be flushed to disk.

3.2 Amortized RRPJ (ARRPJ)

In order to allow RRPJ to be less susceptible to varying data distributions, we introduce Amortized RRPJ (ARRPJ). Suppose there are two partitions P_1 and P_2 , each containing 10 tuples. If P_1 produces 5 and 45 result tuples at

timestamp 1 and 2 respectively, the Th_1 value is 5. If partition P_2 produces 45 and 5 result tuples at timestamp 1 and 2 respectively, the Th_2 value for P_2 will also be 5. From the above example, we can observe that the two scenarios cannot be easily differentiated. However, we should favor partition P_1 since it is obviously producing more results than P_2 currently. This is important because we want to ensure that tuples that are kept in memory are able to produce more results because of its current state, and not due to a past state.

To achieve this, let σ be a user-tunable factor that determines the impact of historical result values. The amortized RRPJ value, denoted as A_i^t , for a partition i at time t is presented in Equation 4. When $\sigma = 1.0$, then the amortized RRPJ is exactly the same as the RRPJ. When $\sigma = 0.0$, then only the latest RRPJ values are considered. By varying the values of σ between 0.0 to 1.0 (inclusive), we can then control the effect of historical RRPJ on the overall flushing behavior of the system.

$$A_i^t = \frac{\sigma^t r_i^0 + \sigma^{t-1} r_i^1 + \sigma^{t-2} r_i^2 + \dots + \sigma^1 r_i^{t-1} + \sigma^0 r_i^t}{N_i} = \frac{\sum_{j=0}^t \sigma^{(t-j)} r_i^j}{N_i} \quad (4)$$

4 Performance Study

In this section, we study the performance of the proposed *RRPJ* against *RPJ*. All the experiments were conducted on a Pentium 4 2.4GHz CPU PC (1GB RAM). We measure the progressiveness of the various flushing policies by measuring the number of results produced and the response time.

Table 1. Experiment Parameters

Dataset Parameter	Default Values
Number of Tuples Per Page	85
Available Memory	1000 pages
Domain of Join attribute	[1, 10000]
Tuple Inter-arrival	0.001s
Dataset Size (Relation R1 + Relation R2)	2 million tuples
Percentage of tuples flushed	10%

The experimental parameters are given in Table II. Unless otherwise stated, the datasets used in the experiments uses the default values given in the table.

4.1 Effect of Uniform Data Within Partitions

We generated the datasets *HARMONY* and *REVERSE* based on the dataset generation techniques described in [6]. We used the same arrival pattern *HARMONY* and *REVERSE*. In this experiment, we evaluate the performance of the RRPJ against RPJ. We measure the response time (x-axis) and the number of

result tuples generated (y-axis). From Figure 2, we can observe that the performance of RRPJ is comparable to RPJ using the same datasets from [6], and hence is at least as effective as RPJ for uniform data.

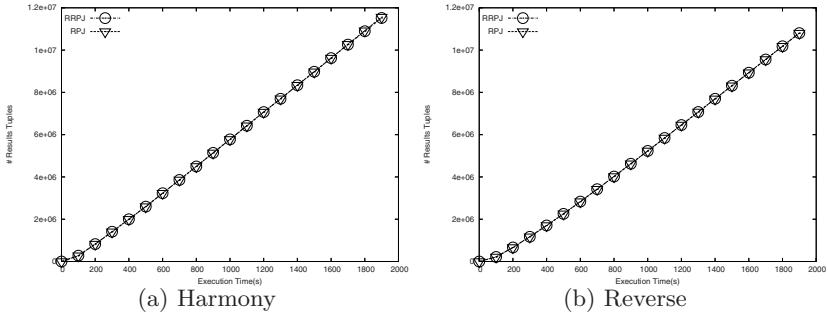


Fig. 2. Effect of Uniform-Data Within Partitions

4.2 Effect of Non-uniform Data Within Partitions

In this experiment, we evaluate the performance of RRPJ against RPJ for non-uniform datasets. We used the same arrival pattern *HARMONY* and *REVERSE*. Similar to Figure 1, we restrict the domain for the join attribute for 50% of the tuples from one dataset (R1) to be in the range [1,5000] and the domain of the join attribute for 50% of the other dataset (R2) to be in the range [5001,10000]. We measure the response time (x-axis) and the number of result tuples generated (y-axis). From Figure 3(a) and 3(b), we can observe that the RRPJ outperforms RPJ by a large margin. This is because RPJ's local uniformity assumption breaks when the data within each partition is non-uniform. Comparatively, since RRPJ tracks the number of results, it is able to identify the partitions that are not producing any results, and hence avoid keeping tuples belonging to these non-productive partitions in memory.

4.3 Varying Data Arrival Distribution

The datasets are generated as follows: We make use of a zipfian distribution (with tunable parameter θ) to determine the partition for assigning a newly-arrived tuple. When $\theta = 0.0$, the data distribution is uniform (i.e. a newly-arrived tuple have equal probability of belonging to any of the partitions). When θ increases, the arrival distribution becomes more skewed (i.e. a newly-arrived tuple have higher probability to belong to specific partitions). In order to simulate a varying data arrival distribution, we re-order the partitions probabilities whenever every α tuples have arrived. The partitions are randomly re-ordered. For example, when $\theta = 2.0$, Table 2 shows the arrival probabilities. During the initial stage, the probability that a newly arrived tuple will belong to partition 1,2,3,4

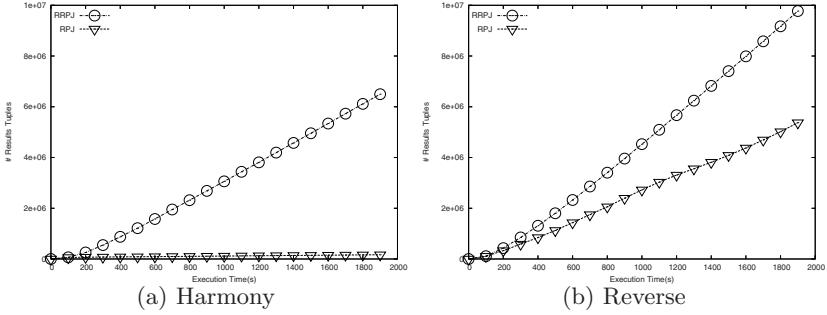


Fig. 3. Effect of Non-Uniform-Data Within Partitions

and 5 are 0.68, 0.17, 0.08, 0.04 and 0.03 respectively. During each reorder, these probabilities for a newly arrived tuple to belong to a specific partition change.

In this experiment, we evaluate the performance of the Amortized RRPJ (ARRPJ) against RPJ and RRPJ, when the data arriving exhibits varying data arrival distribution (i.e the probability that a newly arrived tuple belongs to a partition changes). We vary the amortization factor, σ , for ARRPJ between 0.0 to 1.0. We call the corresponding algorithm ARRPJ- σ . When $\sigma = 0.0$, only the latest RRPJ values (i.e. number of results produced and size of data partition since the last flush) are used; whereas when $\sigma = 1.0$, ARRPJ is exactly RRPJ (it computes the average of the statistics over time).

Table 2. Arrival Probabilities, $\theta = 2.0$

Arrival Probabilities, P	Initial	1st Reorder	2nd Reorder
	Partitions Assigned		
0.68	1	2	3
0.17	2	3	4
0.08	3	4	5
0.04	4	5	1
0.03	5	1	2

The results are shown in Figure 4(a)-(f). In addition, we summarize the throughput (i.e. number of result tuples produced over time) of each algorithm in table 3. In Table 3, we can observe that an amortization factor = 0.0 need not necessarily be the best (highlighted in bold). There is a need to balance between the impact of past and current results. From Figure 4(a)-(e), we can observe that ARRPJ (with different amortization factor) performs much better than RRPJ. Also, when the data distribution changes frequently (e.g. Figure 4(f), $\alpha = 0k$), the performance of RRPJ and ARRPJ are similar.

When $\alpha = 0k$, the data arrival distribution is re-ordered aggressively (changes each time a tuple arrives). Thus, all the methods (including RPJ and XJoin) perform similarly. This is because none of the methods can make use of the

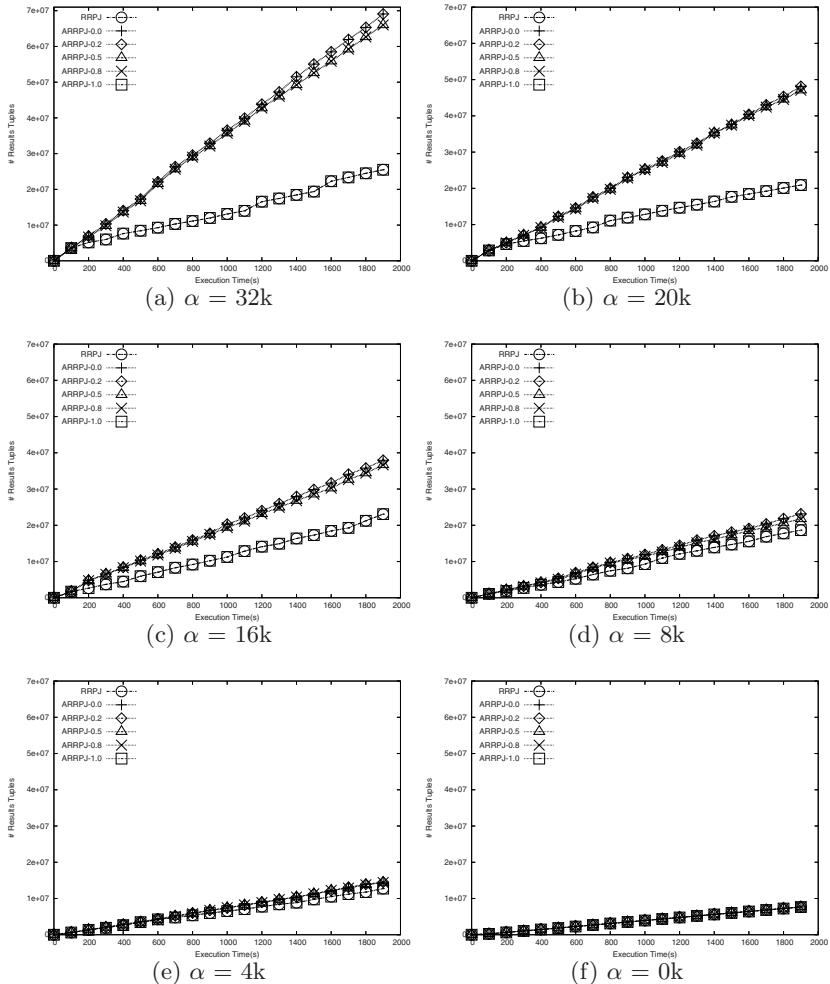


Fig. 4. Varying Data Distribution

Table 3. Throughput of various methods (Summary of Fig 4)

α	RRPJ	ARRPJ-0.0	ARRPJ-0.2	ARRPJ-0.5	ARRPJ-0.8	ARRPJ-1.0
0	4113	4128	4128	4125	4119	4113
4	6735	7719	7950	7665	7541	6735
8	9783	12266	12009	11503	10551	9783
16	11879	20133	20038	19428	17307	11879
20	10027	25140	25152	24554	20887	10027
32	12177	36388	36053	34685	27120	12177

statistics gathered to do effective prediction of which tuples to keep in memory combined with a generally smaller number of possible results. However, when α increases from 4k to 32k, we can observe that ARRPJ (with different α) outperforms RRPJ. This is because ARRPJ was able to better reduce the impact of the past results by amortizing the RRPJ values. RRPJ do not perform as well, since RRPJ does not differentiate between past and current results. From Figure 4, we can also observe that as the data changes less frequently (i.e. when α varies from 0K to 32K), the total number of result tuples significantly increases. This is because when the data distribution changes less often, the statistics computed could be used for more effective prediction of which tuples need to be kept in memory.

In addition, we also varied ρ (percentage of pages flushed each time memory is full, and θ (skewness of the data distribution). Similar trends are observed. When θ is 0.0 (i.e. uniform data), all methods (i.e. RPJ, RRPJ, ARRPJ) performs the same. The results are omitted due to space constraints. These experiments suggest however that several factors influence the correct evaluation of the output statistics when data distribution is changing over time. The amortization formula must be tuned with respect to the size of the buffer, the percentage and size of the replaced partitions as well as the frequency of the replacement. While the purpose of this paper is to introduce the idea of amortization and illustratively quantify its potential, such fine tuning is left to future work.

5 Conclusion

We proposed a new adaptive and progressive equijoin algorithm for relational data streams. The algorithm is of the X- and symmetric hash join family. Its originality is twofold.

Firstly, the algorithm implements a replacement strategy for main memory partitions that estimates the probability of partition to produce results directly from the observation of output statistics. Previous proposals, such as the RPJ and LA algorithms, have attempted to analytically construct such a model from the statistics on the input streams. We showed that our algorithm is equivalent to RPJ in the cases for which RPJs performance was evaluated by its inventors (we use the same data sets). We showed that our algorithm significantly outperforms RPJ, when the uniformity hypothesis necessary to the estimation by the RPJ algorithm does not hold. We therefore showed that our algorithm is globally better than RPJ empirically.

Secondly, we proposed an adaptive version of our algorithm that makes use of amortization in order to incrementally weight out the influence of past statistics. The same principle can be incorporated in previously proposed algorithms such as RPJ and LA. This allows the algorithm to cater for changes over time in the input data distributions. We showed that this technique leads to significant performance increase in some cases, thus proving the concept. However, the results we obtained compel further studies in order to understand the impact of the different parameters. Future and ongoing work includes the practical tuning

of such parameters: amortization formula, buffer size, frequency of replacements and percentage/size of replaced partitions.

Finally we underline that, as we had preliminarily shown in [13], as opposed to RPJ and LA, our approach rather gracefully generalizes to non-relational data as it does not require the complex analytical modeling of the probabilities of partitions to produce results from a model of the input data distribution but rather directly observes a statistical model of the output distribution. We are currently investigating the performance of RRPJ against RPJ and LA in non-relational domains.

Acknowledgments. We would like to thank Dr Tao Yufei and his colleagues for providing us with the RPJ code and data generator.

References

1. Haas, P.J., Hellerstein, J.M.: Ripple join for online aggregation. In: SIGMOD. (1999) 287–298
2. Urhan, T., Franklin, M.J., Amsaleg, L.: Cost based query scrambling for initial delays. In: SIGMOD. (1998) 130–141
3. Urhan, T., Franklin, M.J.: XJoin: Getting fast answers from slow and bursty networks. Technical Report CS-TR-3994, Computer Science Department, University of Maryland (1999)
4. Avnur, R., Hellerstein, J.M.: Eddies: Continuously adaptive query processing. In: SIGMOD. (2000) 261–272
5. Madden, S., Shah, M.A., Hellerstein, J.M., Raman, V.: Continuously adaptive continuous queries over streams. In: SIGMOD. (2002) 49–60
6. Tao, Y., Yiu, M.L., Papadias, D., Hadjieleftheriou, M., Mamoulis, N.: Rpj: Producing fast join results on streams through rate-based optimization. In: SIGMOD. (2005) 371–382
7. Li, F., Chang, C., Kollios, G., Bestavros, A.: Characterizing and exploiting reference locality in data stream applications. In: ICDE. (2006) 81
8. Dittrich, J.P., Seeger, B., Taylor, D.S., Widmayer, P.: Progressive merge join: A generic and non-blocking sort-based join algorithm. In: VLDB. (2002) 299–310
9. Dittrich, J.P., Seeger, B., Taylor, D.S., Widmayer, P.: On producing join results early. In: PODS. (2003) 134–142
10. Mokbel, M.F., Lu, M., Aref, W.G.: Hash-merge join: A non-blocking join algorithm for producing fast and early join results. In: ICDE. (2004) 251–263
11. Lawrence, R.: Early hash join: A configurable algorithm for the efficient and early production of join results. In: VLDB. (2005) 841–852
12. Wilschut, A.N., Apers, P.M.G.: Dataflow query execution in a parallel main-memory environment. In: PDIS. (1991) 68–77
13. Tok, W.H., Bressan, S., Lee, M.L.: Progressive spatial join. In: SSDBM. (2006) 353–358

GChord: Indexing for Multi-Attribute Query in P2P System with Low Maintenance Cost

Minqi Zhou, Rong Zhang, Weining Qian, and Aoying Zhou

Department of Computer Science and Engineering, Fudan University, Shanghai 200433, China
`{zhoumingqi, rongzh, wnqian, ayzhou}@fudan.edu.cn`

Abstract. To provide complex query processing in peer-to-peer systems has attracted much attention in both academic and industrial community. We present GChord, a scalable technique for evaluating queries with multi-attributes. Both exact match and range queries can be handled by GChord. It has advantages over existing methods in that each tuple only needs to be indexed once, while the query efficiency is guaranteed. Thus, index maintenance cost and search efficiency are balanced. Additional optimization techniques further improves the performance of GChord. Extensive experiments are conducted to validate the efficiency of the proposed method.

1 Introduction

Peer-to-peer (P2P) systems provide a new paradigm for information sharing in large-scale distributed environments. Though the success of file sharing applications has proved the potential of P2P-based systems, the limited query operators supported by existing systems prevent their usage in more advanced applications.

Much effort has been devoted to providing fully featured database query processing in P2P systems [1][2][3][4]. There are several differences between query processing for file sharing and database queries. Firstly, the types of data are much more complex in databases than those in file names. Basically, numerical and categorical data types should be supported. Secondly, files are searched via keywords. Keyword search is often implemented by using exact match query. However, for numerical data types, both exact match queries (or *point* queries) and *range* queries should be supported. The last but not the least, user may issue queries with constraints on variant number of attributes for database applications. This last requirement poses additional challenges for database style query processing in P2P systems. Some existing methods, such as VBI-Tree [2], can only support user queries with constraints on all attributes. Some other methods, namely Mercury [3] and MAAN [4], separately index data on each attribute. Though they can support multi-attribute queries with constraints on arbitrary number of attributes, they are not efficient for indexing data with more than three attributes for two reasons. The first one is that the maintenance cost increases with the number of attributes. Another reason is that the selectivity of indexes on one attribute decreases drastically when the number of attributes increases.

We present GChord, a Gray code based Chord, as a new index scheme supporting multi-attribute queries (MAQ) in P2P environment. It distinguishes itself from other methods in the following aspects:

- GChord utilizes the property of one-bit-difference of Gray code to encode numerical data. This encoding scheme, together with the traditional hash-based indexing technique for categorical data, transforms the MAQ problem to multicast problem in a large-scale network. Fully utilizing the finger table links provided by Chord [5], a general purpose P2P overlay network, GChord provides a solid base for indexing data without modification on the underlying overlay network structure. Thus, it provides a convenient solution to work with existing efficient P2P technologies.
- In GChord, each data tuple only needs to be indexed once. Thus, performance of our method does not directly rely on the number of attributes of data. Compared with other Chord-based methods, it is more efficient in terms of maintenance overhead and search performance.
- In addition to the basic indexing and query processing scheme, GChord introduces optimization techniques called *multicast tree clustering* and *index buddy*. The former provides an efficient implementation of multicast in P2P network for the MAQ problem. The latter shows that by consuming a small portion of storage space for caching index entries, GChord outperforms methods with index duplication in terms of storage cost and query processing.

The remainder of this paper is organized as follows. Section 2 is for related work of GChord. After the problem statement given in Section 3, we introduce the basic GChord in detail in Section 4. In Section 5, we present the optimization techniques of GChord. After the experimental result shown in Section 6, Section 7 is for concluding remarks.

2 Related Works

MAQ is widely studied in centralized database systems. One solution of indexing data for MAQ is hB^{II}-tree [6], which is a combination of multi-attribute index hB-tree [7] and abstract index II-tree [8]. hB^{II}-tree achieves low storage cost and efficient point-and range-query processing for various data types and data distribution. However, the different setting of large-scale distributed systems prevents the application of existing technique in centralized systems in P2P systems.

In large-scale P2P systems, distributed hash tables (DHTs), such as Chord [5], Pastry [9], and CAN [10], are widely used. However, it can only support key-word-based search *lookup(key)* and these hash-based methods usually cannot preserve the locality and continuity of data.

The methods supporting MAQ in structured P2P systems can be classified into two categories. The first one introduces traditional tree-structured index scheme into P2P systems. BATON [11] is P2P index structure based on balanced binary tree. BATON* [12] substitute the binary tree in BATON with an m -way tree. These two can well support single dimensional range query. VBI-tree [2] provides a framework for indexing multi-dimensional data in P2P systems with hierarchical tree-structured index in centralized systems. However, these structures cannot support queries with constraints on arbitrary number of attributes efficiently.

The other category of research work is based on extending DHT-based overlay networks. The basic idea behind these methods is to use one overlay network for each

attribute that needs to be indexed, and to use locality-preserving hash to index numerical attributes. Mercury [3] and MAAN [4] belong to this category. Both of them index each attribute separately on Chord ring and the index with the best selectivity power is used to prune the search to support MAQ. Therefore, both of them have high storage cost and index maintenance cost. Furthermore, the search efficiency decreases drastically when the dimensionality of data increases.

3 Problem Statements

A tuple of data is represented by a set attribute-value pairs: $t\{attr_i, v_i\}, i = 1, \dots, N$. The domain A^i of attribute $attr_i$ is either numerical or categorical. A numerical domain is supposed to be continuous or sectionally continuous, and bounded. Given a data set, the set of domains $\mathbb{A} : \{A^i\}$ for $i = 1, 2, \dots, N$ is supposed to be known in advance. We believe that even with this assumption, many applications can be fit into our MAQ model.

A multi-attribute query (MAQ) is a conjunction of a set of predicates of the form $(attr_i, op, v_i), i = 1, \dots, m$, in which, $attr_i$ is the attribute name, op is one of $<$, \leq , \geq , $=$ for numerical attributes and $=$ for categorical ones. Note that a query may have arbitrary number of predicates, and the predicates may be on arbitrary attributes. Figure 1 shows a simple example of data and queries.

film	name(c)	price(n)	duration(n)	premiere(n)	cinema(c)
Lord of War		50	90	05-06-1	Yongle
Garfield		60	75	06-07-12	Guang
XMan		90	100	05-11-4	Yonghua
Thinking		100	125	06-05-09	Yonghua
The Break		90	110	05-12-17	Yonghua
Silther		90	100	06-10-30	Guang

Records

Q1: fn="Lord of War" \wedge price > 40 \wedge price < 80
Q2: premiere > 06-05-4 \wedge premiere < 06-07-09 \wedge cinema = "Yongle"
Q3: duration > 80 \wedge price < 90 \wedge cinema = "Guang" \wedge premiere > 06-06-17
Q3:fn="XMan" \wedge price > 70 \wedge premiere > 06-03-12 \wedge cinema = "Guang"
Q4:fn="Thinking" \wedge price = 50 \wedge premiere > 06-07-05

Queries

Fig. 1. Data Item and Query

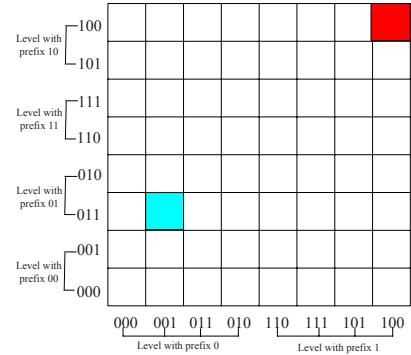


Fig. 2. Two Attributes Domain Partition

The results to a MAQ are the set of tuples satisfying all predicates presented in the query. Note that there is no constraints on the values of attributes missed in the query.

The problem of MAQ in a P2P environment is that a set of peers each may share (or publish) a set of tuples. Each peer may issue a MAQ to be evaluated in a P2P style, which means there is no centralized server, and the peers work in a collaborative mechanism. To collaborate with others, a peer devotes some storage space for indexing data shared by others, and supports index lookup and maintenance.

The difficulty of query processing for MAQ in P2P systems lies in the following three aspects: 1) one query may involve attributes with different data types, and point

constraint and range constraint may appear simultaneously in one query, 2) arbitrary number of attributes may presented in a query, and 3) index maintenance is especially expensive in P2P systems. Since there are $N! - 1$ possible combinations of attributes in a query, any method using index structure that can only answer queries with fixed number of attributes will fail to handle MAQ in P2P environment, for the high cost of index maintenance in distributed environments.

In the next section, we present GChord, a Gray code based indexing scheme that can be distributed over Chord like overlay network. By fully utilizing the network links provided by the underlying network, it indexes each tuple only once, and can support MAQ with arbitrary number of attributes using the sole index.

4 The Basic GChord Mechanism

4.1 Data Indexing

Each attribute is assigned a set of bits in the 128-bit bitstring to store its *code*. The number of bits of a code is proportional to the size of the domain. Note that it is assumed all domains are known in advance. Intuitively, the larger a domain is, the stronger is the selectivity power of that attribute. Thus, more bits should be devoted to index that attribute. Numerical and categorical attributes are encoded differently in GChord.

Numerical and categorical attributes are encoded differently in GChord.

Encoding Numerical Attributes. As the domain of each numerical attribute is pre-defined, GChord partitions the domain equally and continuously. For those sectionally continuous attribute domains, it concatenates all the sections together first, and then makes equally partition among them. For example, if one attribute domain is composed of three continuous sections, $(1, 4)$, $(5, 17)$, $(31, 36)$, the four equally partitioned parts are $\{(1, 4), (5, 7]\}$, $\{(7, 12]\}$, $\{(12, 17]\}$ and $\{(31, 36]\}$. Obviously, each part contains a interzone with same length.

All partitioned parts of one attribute domain are encoded by *Standard Binary Reflected Gray Code* [12] (Gray code for short) continuously and sequently, as Figure 2 shows. Obviously, the two Gray codes that represent two adjacent partition parts differ in one bit. To make Gray code be fully used, we restrict the number of partitioned parts to be 2^k , where k is the number of bits in Gray code.

It's hard to compute the Gray Code from attribute value directly, while it is easy to compute the sequence number of the partitioned part that attribute value fills in by simply using equation, $SN(v) = \frac{(v - v_{j\min}) \times (2^m - 1)}{\sum_{i=0}^n (v_{i\max} - v_{i\min})}$, where v is the attribute value and $v \in [v_{j\min}, v_{j\max}]$, and m is the Gray Code length. The corresponding Gray Code which is the sub index key on the attribute is converted from the sequence number using algorithm 1.

Encoding Categorical Attributes. Since only exact match query is to be supported for categorical attributes in MAQ, it is much easier to encode categorical attributes. A hash-based method is used to determine the code of a categorical value, like $code(v) = hash(v) \bmod 2^m$ in which m is the number of bits assigned to the categorical attribute, and $hash$ is a general purpose hash function such as SHA-1.

Algorithm 1: SN2GC(*BitString sequencenumber[n]*)

```

1 BitString graycode[n] //output Gray code which contains n bits
2 graycode[0]← sequencenumber[0]
3 for each i from 1 to n – 1 do
4   if sequencenumber[i] = sequencenumber[i-1] then
5     |   graycode[i]← 0
6   else
7     |   graycode[i]← 1
8 return graycode

```

Generating the Index Key for a Tuple. As the number of peers that participant in the network is much less than the number of peers that network can accommodate, one peer in the network have to manage many index keys. If the index key is simply constructed by concatenating all N codes, the attribute encoded at the right side will lose its distinguishability. All values of the attribute that is encoded on the right side may be mapped to the same peer. It results in the index on that attribute useless.

GChord provides a *shuffling-based* method to generate the index key of a tuple. The shuffled index key is constructed by concatenating a bit of code of one attribute by that of another. The order of the attributes is pre-determined using the descending order of the size of the domains.

Analysis to the Index Key Generation Method. Since two adjacent Gray codes only differs in one bit, the adjacent relationships between two sections of numerical attributes are preserved by any structured overlay network protocols that maintains one-bit different links in routing tables, such as Chord and Pastry.

Property 1. *Two index keys have one bit difference if the two corresponding tuples have adjacent values on one numerical attribute and same values on other attributes.*

Thus, our indexing scheme preserves locality of numerical attributes.

Property 2. *The index keys stored on one peer are constituted by a set of continuous partitions on each numerical attribute.*

Property ① means adjacent values on each attribute are linked by the links in routing table of the overlay network like Chord and Pastry. Query message can be routed efficiently for index keys of adjacent data are one hop away. Property ② means that part of the queried region may addressed by accessing one peer. As the overlay is not fully filled, routing hops may be saved if index keys are inserted into the predecessor.

As the index key is shuffled, load balancing can achieved simultaneously. Since the distribution of real data are always skewed and the sections of attribute domain are encoded by equally partition, the data tuple filled in one section may skewed. Some new strategies need to adopt to keep load balancing among peers in the network.

Lemma 1. *The prefixes of a set of Gray codes, which have a same bit length, construct the Gray Code sequence either.*

Property 3. *The index keys stored on each peer constitute a similar size portion on each attribute.*

Property 4. *The process of node join is the process of the attribute domain repartition.*

As known from Property 4, load balancing can be achieved by selecting some suitable ID for node at the time of node join.

4.2 Query Processing

To evaluate a MAQ, a set of nodes with index entries satisfying the query should be visited. Intuitively, attributes presented in the MAQ should satisfy the predicates in the query. Therefore, their corresponding bits in the peer identifier, i.e. the codes of those attributes, should satisfy the constraints. There are no constraints on other bits. Thus, the query processing procedure can be transformed into a multicast problem. The targeted peers are peers taking care of the identifiers satisfying the following constraints:

1. For each predicate $attr \ op \ v$ on a numerical attribute $attr$, $code(attr) \ op' \ code(v)$;
2. For each predicate $attr = v$ on a categorical attribute $attr$, $code(attr) = code(v)$;
3. All other bits can be either 0 or 1.

Thus, a multicast task can be represented by a set of strings with the same length as that of the identifier (index key). Each element in the string is either 0, 1 or x , in which x means either 0 or 1.

Multicast trees (MCTs) are constructed to forward the query to indexing peers. A MCT is a virtual tree whose edges are routing paths of the query. A MCT corresponding to the multicast of $10xx1xx$ is shown in Fig. 3.

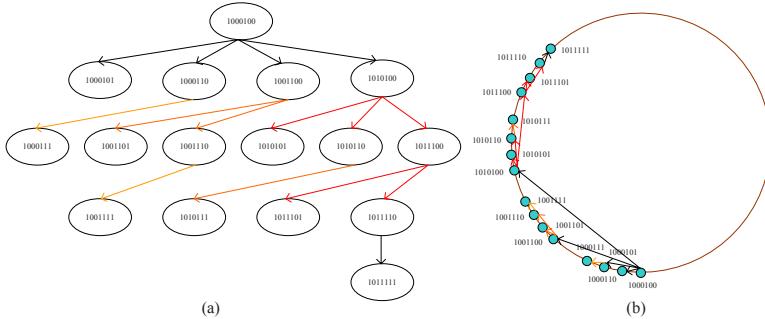


Fig. 3. Multicast Tree of $10xx1xx$

Multicast Tree Construction. As the links in the finger table of overlay network are directed, one single MCT without irrelevant indexing nodes for MAQ may not exist. The MCTs should be constructed on-the-fly when a query is issued. A modified Karnaugh Map [13] construction algorithm is employed for this task.

Karnaugh Map is a graphical way of minimizing a boolean expression based on the rule of complementation. Preventing processing a Karnaugh Map of large size, we compute *Multicast Tree Proportion* (MTP) of two attributes at each time. We need compute $\lceil \frac{m}{2} \rceil$ times to get all the MTPs. If m is odd, the last *MTP* is computed on single attribute. Attribute that is not present in the query has a MTP in the form of “ $xx \dots x$ ” which has a same length with the code represented the attribute partition. After all MTPs are computed, they are shuffled and put together using the method we generate the index keys.

Supposing the number of MTPs on each attribute that contains constraints in the MAQ is n_1, n_2, \dots, n_m , the number of MTPs of the MAQ is $n_1 \times n_2 \times \dots \times n_m$. The procedure to compute MTP is as follows: (1)Initialize an empty Karnaugh Map: each side of the map has the length equal to the length of the code of the corresponding attribute. (2)For the cells satisfying all constraints given by the predicates on attributes presented in both sides, they are marked by “1”. All other cells are marked by “0”. Each cell forms a rectangle of size 1. (3)Two adjacent rectangles containing cells all marked by “1” are merged to generate a rectangle with size 2^i . Note that Karnaugh Map is a torus. Thus a leftmost cell and a rightmost cell in the same column are considered adjacent. And likewise is to the top and bottom cells. (4)The step 3 iterates until there is no larger rectangles can be generated.

Fig. 4 shows a Karnaugh Map with three MTPs corresponding to multicast tasks $< x0x, x0x >$, $< x1x, 11x >$, and $< x1x, 101 >$.

a \ b	000	001	011	010	110	111	101	100
000	1	1	0	0	0	0	1	1
001	0	0	0	0	0	0	0	0
011	0	0	0	0	0	0	0	0
010	0	0	0	0	0	0	0	0
110	0	0	1	1	1	1	0	0
111	0	0	1	1	1	1	0	0
101	0	0	1	1	1	1	0	0
100	1	1	0	0	0	0	1	1

Fig. 4. Karnaugh Map on Two Attributes

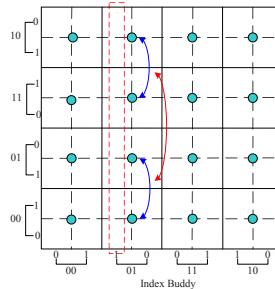


Fig. 5. Index Buddy

Property 5. Supposing query region on two attributes is a $A \times B$ rectangle, where A and B are the numbers of partition parts contained in query. Supposing the binary forms of A and B contains m, n “1”s respectively. The query rectangle can be divided into $\frac{m^2+n^2+3m+n}{2} + 1$ (if $m \geq n$) MTPs.

Proof: As the number of cells which contained in the MTP rectangle must be power of two, the number of cells on each side of the rectangle must be power of two either. Obviously the A cells on one attribute is divided into m parts. The B cells on the other side of Karnaugh Map is divided into n parts. After once division, we get $m + n - 1$ MTPs and one $(A - 2^{a_1}) \times (B - 2^{b_1})$ rectangle that haven't been divided, where a_1 and b_1 are the position of first “1” in A 's and B 's binary form. Then we do it recursively. \square

After all the MTPs having been generated, we could shuffle these MTPs on each attribute like constructing index key to construct the MCT.

Our Karnaugh Map based MCT construction techniques can be generalized to handle multiple queries. Targeted index peers from different MAQs may be grouped together into one MCT using the same technique introduced above. Thus, the queries may share the same message to retrieve the index entries. This may further reduce the network transmitting cost.

Query Multicasting. After all the MCTs corresponding to the query have been generated, multicasting of a query is conducted as follows: (1) Query message is sent to the root of each MCT which is a peer with identifier by substituting all xs in the MCT representation with 0s. (2) When a query is received by a peer, it is evaluated on its local index, and forwarded to all peers whose identifier is substituting one of the xs in MCT representation from 0 to 1. (3) This is conducted recursively until there is no x remains 0.

Fig. 3(b) illustrates a multicast process.

Property 6. *The query message routing hops can be bounded to $O(\log_2 N + M)$, where N is the number of nodes that overlay network can accommodate and M is the number of xs in the MTP representation.*

5 Performance Enhancement

The number of attributes which contain constraints in query could vary from 1 to N . More MCTs will be generated, if there are more range constraints on attributes contained in query. The number of MCTs is a product of the number of MTPs on each attribute. The cost of multicasting a large number of MCTs involved in the query separately is very high.

On the other hand, the query range will be very large if there are fewer attributes which contain constraints in the query. A large number of peers have to be accessed to process such MAQ. If MAQ is addressed by accessing a large number of peers, the number of query message routing hops and query messages will be high. In the two scenarios above, performance can be enhanced by *multicast tree clustering* and *index buddy* respectively.

5.1 Multicast Tree Clustering

Peers scattered on the overlay network are sparse, so each peer needs to manage a set of continuous index keys. A portion of continuous results to the query may be indexed on the single peer, but these index keys may be accessed by different MCTs for constraints of MCT computing strategy. In this scenario, a number of messages have to be sent to the same peer to get result for the query. If these MCTs are clustered together, and sent within single message, lots of network traffic will be saved.

MCT clustering strategy clusters MCTs which are close to each other together. Before doing MCT clustering, peers in the network need to know the approximately peer density of overlay network. Avoiding network traffic, we estimate the peer density using *local density* which is the reciprocal of the index key range that peer maintained. No matter how inaccurate the density estimation is, it has no impact on query results.

MCT clustering only sends multi-MCTs in single query message, but it does not affect the query evaluation to each MCT.

The clustering procedure is as follows: (1)Query submitter clusters all the MCTs together, which contain close root keys. Namely the difference of each two root keys is less than the index range that maintained by the peer. (2)Query message is sent according to the root key which is thesmallest one within the MCT cluster. (3)Peer received the MCT cluster clusters all adjacent submulticast trees together as query submitter does. (4)Procedure 3 is done recursively until no sub multicast trees exists.

As many MCTs and sub MCTs are sent within one message, the number of message for one query is reduced dramatically.

5.2 Index Buddy

If MAQ contains a large query region, a large number of peers will be involved to process the query. The response time and network bandwidth consumption will be enlarged if more peers are involved. Avoiding to involve too many peers, we adopt index buddy strategy. An index buddy is tow peers which store adjacent values on the same attribute and have a same peer Id prefix.

Users may have similar interests at the same time. For example users may submit similar queries to get match list during the time of Olympic Games. If these frequently queried index keys are stored on a few peers, reduced number of routing hops and query messages can be achieved.

As described in Property 2, index keys stored on peer are continuous on each attribute. Adding index keys that is frequently queried to the peer maintains adjacent values on the attribute, MAQ can be addressed by accessing fewer peers in the network. Fig. 5 shows the index buddy. The frequently queried region is depicted within the red rectangle. Peers which maintain index keys within this region will manage the index keys of its index buddy's either. Obviously, half of the peers can be released from processing the MAQ. The procedure of doing index buddy is as follows:

- **Partition Level Sampling.** Before exchanging index keys between index buddies, we need to know the range of index keys of each attribute stored on the buddies. We compute the *index range* using $IR = succ.id - id$. The number of partitions managed by the peer is 2^{k_i} on the i th attribute, where k_i is the number of bits used to represent partitions on the attribute in IR 's binary form.
- **Frequent Query Region Detecting.** We maintain one counter to indicate the current query frequency for each attribute. The counter is a fade function that records the number of messages which are relayed to get the adjacent value on the very attribute through links in the finger table. We use Equation $FQA_{new}^i = \frac{FQA_{old}^i}{2} + \frac{N_{new}^i}{T_{time}}$ to estimate the query frequency of the i th attribute, where N_{new} the number of messages that are sent to get the adjacent index keys on the i th attribute, T_{time} is a specified interval time. When the frequency of the i th attribute exceed the threshold $FQA_{threshold}^i$, the region of the attribute stored on the peer is regarded as frequently queried.
- **Index Buddy Establishing and Deleting.** When detecting some attribute region stored on the peer is frequently queried, the peer asks its index buddy to exchange

their index keys within the region. When detecting region becomes infrequent again, the two peers will remove these redundant index keys out of the index buddy.

- **Index Modification.** When index buddy existing, new index to be inserted or existing index to be modified are need to be processed at both site of the index buddy in order to keep index consistent.

6 Experimental Study

To evaluate the performance of GChord, we implement one simulator in Java JDK 1.42. In our implementation, each peer is identified by its peer Id. Like physical peer, it maintains two limited message queues, one sending message queue and one receiving message queue. The network layer is simulated to control the network communication, which is the message sending from one peer to another based on peer Ids.

In our experiment, 10000 peers with randomly distributed peer Ids are involved to construct the Chord ring. The peer Id is a 32-bit string. The data tuple contains 5 numerical attributes and 1 categorical attribute. 100000 data tuples with randomly distributed values within their attribute domains need to be indexed. Range queries which have been set maximum query range are generated randomly within the attribute domains. Point query is generated randomly within the attribute domains.

Impact of Attribute Number in MAQ. The first set of experiments gives the performance curves impacted by variable number of attributes which contain constraints in the query. The maximum query range on each attribute is set to be 10% of its domain. As showing in Fig. 6(a), 6(b) and 6(c), the numbers of maximum routing hops, routing messages and accessed peers reduce dramatically when the number of attributes that contain constraints in MAQ increase. The number of routing messages reduces to about one tenth when using multicast tree clustering strategy. Multicast tree clustering improves performance pretty well especially when query contains fewer attributes.

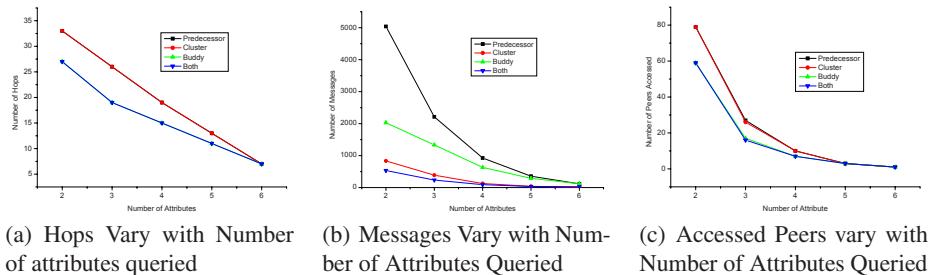


Fig. 6. Performance Comparison with Variable Attributes in Query

Impact of Query Range in MAQ. In this set of experiments, the number of attributes that contain constraints in query is set to be 4. As showing in Fig. 7(a), 7(b), and 7(c), the numbers of maximum routing hops, routing messages and accessed peers decrease as except when the query range on each attribute decreases. More MCTs will be generated, when the query range on each attribute increases. Much more routing messages are diminished by using multicast tree cluster in this scenario.

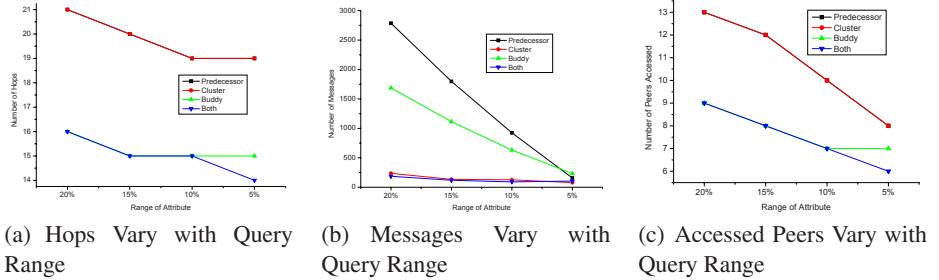


Fig. 7. Performance Comparison with Variable Query Range

Impact of Frequently Queried Region. In this set of experiments, the maximum query range on each attribute is set to be 10% of its domain. As showing in Fig. 8(a), 8(b) and 8(c), index buddy has evident effort in reducing the number of peers accessed when the percentage of frequent query increase. Index buddy has a similar impact on the maximum number of routing hops, especially when query contains less attributes.

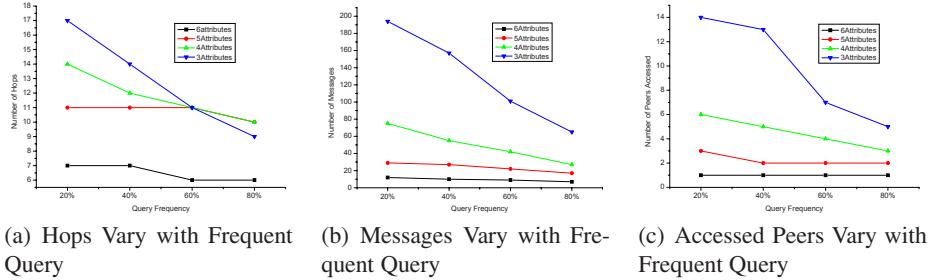


Fig. 8. Performance Comparison with Frequent Queries

Comparison with Mercury. As there are 10000 peers in the network, the number of maximum hops and accessed peers in Mercury is much bigger than GChord's. Approximately 1700 peers construct a Chord ring to maintain the index keys on each attribute. The selectivity power of the attribute is very strong, so Mercury need to accessed a large number of peers to process MAQ. Index keys are stored continuously on peers, so accessing adjacent index key need only one more hop. That's why the number of routing

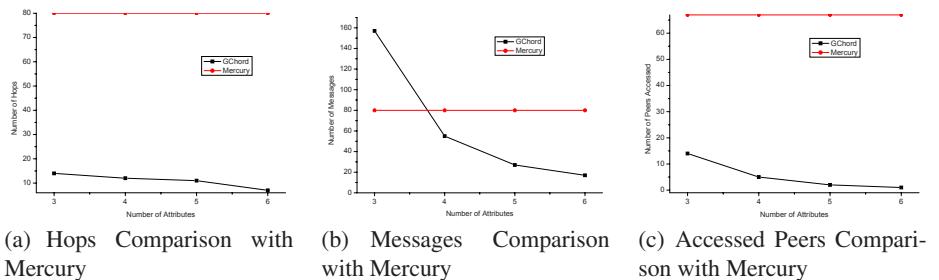


Fig. 9. Performance Comparison with Mercury

messages is smaller than GChord's. As showing in Fig. 9(a) and 9(c), the performance of the GChord exceeds Mercury much in the number of maximum routing hops and accessed peers.

As the limitation of paper size, the comparison of index cost with Mercury is no showing in figures. Mercury keeps one index duplication for each attribute, so the index cost of Mercury is proportional to the number of attributes that data tuple contains. So the index costs of GChord, including index storage and index messages, are much less than Mercury's.

7 Conclusion

In this paper, we present the design of GChord, a P2P-based indexing scheme for processing multi-attribute queries. Using Gray code based indexing technique, both point-and range-query on numerical attributes can be handled. By integrating Gray code and hash based encoding method, each tuple only need to be indexed once in GChord. Our index can support queries having constraints on arbitrary number of attributes. Thus, it is more efficient than previous methods in terms of storage cost and search performance. Enhancement techniques further improves the performance of GChord.

Our future work on GChord includes the research on supporting keyword-based queries and aggregate queries over GChord, and the study on more intelligent query optimization techniques.

References

1. H.Jagadish, B.Ooi, Q.Vu: Baton: A balanced tree structure for peer-to-peer networks. In: VLDB. (2005)
2. H.Jagadish, B.Ooi, Q.Vu, R.Zhang, A.Zhou: Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In: ICDE. (2006)
3. R.Bharambe, M.Agrawal, S.Seshan: Mercury:supporting scalable multi-attribute range queries. In: SIGCOMM. (2004)
4. M.Cai, M.Frank, J.Chen, P.Szekely: Maan:a multi-attribute addressable network for grid information services. In: Grid. (2003)
5. I.Stoica, R.Morris, D.Karger, F.Kaashoek, H.Bhalakrishnan: Chord: A scalable peer-to-peer lookup service for internet applications. In: ACM SIGCOMM. (2001) 149–160
6. G.Evangelidis, D.Lomet, B.Salzberg: The hbpi-tree: a multi-attribute index supporting concurrency,recovery and node consolidation. VLDB Journal **6** (1997) 1–25
7. D.Lomet, B.Salzberg: The hb-tree: a multiattribute indexing method with good guaranteed performance. ACM Trans Database Syst **15** (1990) 625–658
8. D.Lometand, B.Salzberg: Access method concurrency with recovery. In: SIGMOD. (1992)
9. A.Rowstron, P.Druschel: Pastry:scalable,decentralized object location and routing for large-scale peer-to-peer systems. In: Middleware. (2001) 329–350
10. S.Francis, P.Handley, M.Karp, R.Shenker: A scalable content-addressable network. In: SIGCOMM. (2001)
11. H.Jagadish, B.Ooi, K.LeeTan, Q.Vu, R.Zhang: Speeding up search in peertopeer networks with a multiway tree structure. In: SIGMOD. (2006)
12. F.Gray: Pulse code communications. In: U.S. Patent 2632058. (1953)
13. M.Karnaugh: The map method for synthesis of combinational logic circuits. AIEE **72** (1953) 593–599

ITREKS: Keyword Search over Relational Database by Indexing Tuple Relationship

Jiang Zhan and Shan Wang

School of Information, Renmin University of China, Beijing, 100872, P.R. China
{zhanjiang, swang}@ruc.edu.cn

Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), MOE, Beijing 100872, P.R. China

Abstract. Keyword-based search is well studied in the world of text documents and Internet search engines. While traditional database management systems offer powerful query languages, they do not allow keyword-based search. In this paper, we discussed ITREKS, a system that support efficient keyword-based search over relational database by indexing tuple relationship: A basic database tuple relationship, FDJT, is established in advance. Then a FDJT-Tuple-Index table is created, which records relationships between each tuple and FDJT. At query time, for each of keywords, system first finds tuples in every relation that contain it, using full text indexes offered by database management system. Then use FDJT-Tuple-Index table to find the joinable tuples contain all keywords in the query.

Keywords: keyword search, relational database, full disjunction.

1 Introduction

Keyword-based search is well studied in the world of text documents and Internet search engines, but Keyword-based search over relational databases is not well supported. The user of a relational database needs to know the schema of the database; Casual users must learn SQL and know the schema of the underlying data even to pose simple searches. For example, suppose we have a DBLP database, whose schema is shown in Figure 1. We wish to search for an author Bob's paper related to "relation". To answer this query, we must know how to join the Author, Write and Paper relations on the appropriate attributes, and we must know which relations and attributes to contain "Bob" and "relation". In keyword-based search, for the above example a user should be able to enter the keywords 'Bob relation' and the associated tuples which are associated with the two keywords are returned.

Enabling keyword search in databases that does not require knowledge of the schema is a challenging task. Due to database normalization, logical units of information may be fragmented and scattered across several physical tables. Given a set of keywords, a matching result may need to be obtained by joining several tables on the fly.

In this paper, we have developed a system, ITREKS (Indexing Tuple Relationship for Efficient Keyword Search), which supports highly efficient keyword-based search

over relational databases by indexing tuple relationship. The key features and advantages of our approach, and the contributions of this paper, are summarized as follows:

- Most previous approaches perform a significant amount of database computation at search time to find the connection of tuples which contain keyword. We do all significant join computing work in advance by create tuple relation index, so a great amount of computing work is saved in search time.
- We present a novel approach to index the tuple relationship. We construct basic tuple relationship-FDJT by computing full disjunction[1] of the interconnected relational database. We present an FDJT-Tuple-Index table to index tuples' relationship.
- We propose a modular architecture and have implemented ITREKS based on it.
- We present an efficient algorithm which incorporate basic tuples and FDJT-Tuple-Index table to generate result tuples matching the query.
- We take full advantage of existing relational database functions. ITREKS has been implemented on top of Oracle 9i. Oracle 9i Text use standard SQL to create full text indexes on text attributes of relations. We completely avoid reimplementing basic IR capabilities by using Oracle Text as the back end. Furthermore, ITREKS keep both FDJT and tuple-FDJT in relation tables. Our searching algorithm is also based on a search table.

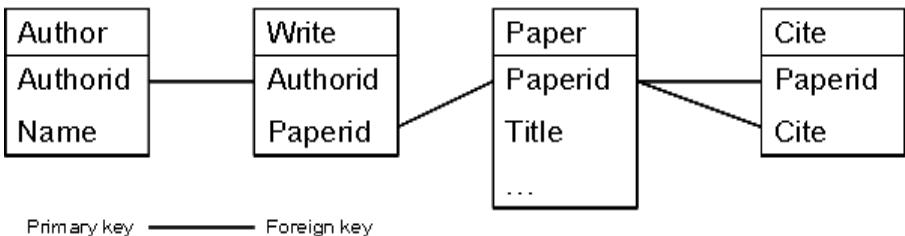


Fig. 1. DBLP Schema

In Section 2 we provide a thorough survey of related work. The essential formal background on full disjunction and related definition is presented in Section 3. Section 4 is the core of the paper. It discusses the ITREKS system, including architecture of the system, functionality, algorithms, and system implementation details. Section 5 presents our system evaluation of ITREKS, while we give conclusion and future works in Section 6.

2 Related Work

Oracle [3], IBM DB2, Microsoft SQL Server, PostgreSQL, and MySQL all provide text search engine extensions that are tightly coupled with the database engine. However, in all cases each text index is designed over a single column. Using this feature alone to do meaningful keyword search over an interconnected database would require merging the results from many column text indexes.

Keyword-based search over relational database gets much attention recently. Three systems, DISCOVER[4] [5], BANKS[6], and DBXplorer[7], share a similar approach: At query time, given a set of keywords, first find tuples in each relation that contain at least one of the keywords, usually using database system auxiliary full text indexes. Then use graph-based approaches to find tuples among those from the previous step that can be joined together, such that the joined tuple contains all keywords in the query. All three systems use foreign-key relationships as edges in the graph, and point out that their approach could be extended to more general join conditions. A main shortage of the three systems is they spend a plenty of time to find the candidate tuples that can be joined together.

Four systems share the concept of crawling databases to build external indexes. Verity[8] crawls the content of relational databases and builds an external text index for keyword searches, as well as external auxiliary indexes to enable parametric searches. DataSpot[9] extracts database content and builds an external, graph-based representation called a hyperbase to support keyword search. Graph nodes represent data objects such as relations, tuples, and attribute values. Query answers are connected subgraphs of the hyperbase whose nodes contain all of the query keywords. DbSurfer[10] indexes the textual content of each relational tuple as a virtual web page. Given a keyword query, the system query and navigate the virtual web pages and find the results. EKSO[11] indexes interconnected textual content in relational databases, and do keyword search over this content. A relational database is crawled in advance, text-indexing virtual documents that correspond to interconnected database content. At query time, the text index supports keyword-based searches with interactive response, identifying database objects corresponding to the virtual documents matching the query.

All the index-data-offline systems have two challenges, how to control the granularity of the indexed content and how to efficiently find the exact results from the indexed content.

While a direct empirical comparison between our system and some of the other approaches mentioned in this section would be very interesting, the comparison is not feasible for the follow reasons:

- The systems are not publicly available.
- The systems implemented different search semantic and different result sets.
- Any effort to implement them well enough for a fair comparison would be prohibitive.

3 Background

3.1 Basic Tuple Relationship

In our method, we need first to find the closest and the most important connection among tuples. In general, if we have any collection of facts that agree on common attributes (are *join-consistent*) we would like them to be available in the “result” of this collection of facts. The problem is related to that of computing the full outerjoin of many relations in a way that preserves all possible connections among facts. Such a computation has been termed a “full disjunction” by Galindo-Legaria[1]. A full disjunction is a relation with nulls (represented by \perp) such that every set of

join-consistent tuples in our database appears within a tuple of the full disjunction, with either \perp or a concrete value in each attribute not found among our set of tuples. Each tuple of full disjunction is corresponding to a set of connective tuples, each of them from a database relation. Naturally, full disjunction reflects the closest and most important relationship among the tuples that generate them. Through full disjunction, we can build the basic relationship of the tuples that come from different database relation.

3.2 Full Disjunction

We consider a database that has n relations R_1, \dots, R_n . The schema graph G is an undirected graph that captures the primary key to foreign key relationships in the database schema. It has a node R_i for each relation R_i of the database and an edge from R_i to R_j for each primary key to foreign key relationship. We assume that schema graph G is connected, which is a reasonable assumption for realistic database schema design.

Definition 1 (Tuple Subsumption). We say that tuple t subsumes tuple u if t and u agree in every component where u is not \perp . That is, t is obtained from u by replacing zero or more nulls by concrete values. Note that a tuple t subsumes itself.

Definition 2 (Full Disjunction). Let $\Gamma = R_1, R_2, \dots, R_n$ be relations whose tuples do not have nulls. We say R is the **full disjunction** for Γ if the following hold:

- No redundancy: No tuple of R subsumes any other tuple of R .
- Tuples of R come from connected pieces of Γ : Let t be a tuple of R . Then there is some connected subset of the relations of Γ such that t , restricted to its non null components, is the join of tuples from those relations.
- All connections are represented:
 - Let t_1, \dots, t_k be tuples chosen from distinct relations R_{i_1}, \dots, R_{i_k} , respectively, such that the schema graph of $\{R_{i_1}, \dots, R_{i_k}\}$ is connected.
 - Let the t_i 's be join-consistent, in the sense that for any attribute A , all the components among the t_i 's corresponding to attribute A have the same value.
 - Let t be the tuple that agree with each of the t_i 's in those attributes appearing among any of R_{i_1}, \dots, R_{i_k} and that has \perp in other attributes found among the schemes of Γ .

Then t is subsumed by some tuple of R .

Definition 3 (FDJT). Let t be a full disjunction tuple. We call such tuples t_1, \dots, t_k **Full Disjunction Join Tuples (FDJT)** of t , if t can be generated by joining the set of tuples t_1, \dots, t_k . Each full disjunction tuple is corresponding to a set of tuples like t_1, \dots, t_k . Note that tuples in FDJT doesn't have sequence.

Definition 4 (FDJTR). **Full Disjunction Join Tuples Relation** is a relation made up with FDJTs of all tuples in full disjunction. In ITREKS, FDJTR is made up with relation names and tupleIDs (or rowids) of tuples in FDJT. Each pairs of relation

name and tupleID represent a database tuple of a FDJT.

It was proved that full disjunction is unique^[2]. It is easy to prove that FDJT is also unique; otherwise there will be two equivalent tuples in full disjunction.

3.3 Computing Full Disjunction and FDJTR

We would like to find a simple way of computing the full disjunction of a set of relations. The solution is to compute full disjunction by full outerjoin. The full outerjoin is a variant of the join in which tuples of one relation that do not match any tuple of the other relation are add to the result, padded with nulls. This operation is part of the SQL92 standard. This problem of computing full disjunction by outerjoin was studied by Galindo-Legaria in [1]. [1] gave a test for when some order of outerjoins is guaranteed to produce the full disjunction by itself. This test is simple. Create a graph whose nodes are the relations and whose edges connect relations that are constrained by one or more comparison; if the graph is acyclic then the full disjunction can be computed applying full outerjoins in any order. For cyclic graphs, however, the full disjunctions don't exist. Thus we have the Lemma 1.

Lemma 1. For a database which has an acyclic connected scheme graph, we can compute full disjunction by applying full outerjoin of the connected relations in any sequence.

Now for a database whose scheme graph is acyclic, we can use Lemma 1 to generate a full outerjoin sequence producing the full disjunction. In the above full outerjoin sequence, each relation appears exactly once. The relation tuples which are outerjoined to generate a tuple of full disjunction is FDJT of this tuple. Algorithm 1 generates FDJTR when computing full disjunction of a database.

Algorithm 1. Computing FDJTR

Input: database relations R_1, \dots, R_n , connected acyclic database graph G

Output: FDJTR of the database

1. Do breath-first traversal on G from one of the G's leaf node of G, get a sequence of the relations R'_1, \dots, R'_n .
 2. Let FD_k store full disjunction of some connected relations R'_1, \dots, R'_k ($k=2$ to n), F_k store FDJTR of FD_k .
 3. $FD_1 = R'_1$
 4. Add R'_1 to F_1
 5. **for** $i=2$ to n
 6. $FD_i \leftarrow FD_{i-1}$ full outerjoin R'_i
 7. **foreach** tuple t in FD_i , add FDJT of t to F_i
 8. **end for**
 9. **return** F_i
-

Algorithm 1 first generates the relations sequence by breath-first traversal over G, then full outerjoins the relations in turn, computing the full disjunction and corresponding FDJTR of the relations.

We will discuss how to compute FDJTR of database whose schema is cyclic in Section 4.

4 The ITREKS System

The system we have developed, ITREKS, is an instantiation of the general architecture we propose for keyword search over databases that is shown in Figure 2. Given a set of query keywords, ITREKS returns all *results* (sets of joinable tuples from relations connected by foreign-key relationship) such that each result contains *all* keywords. Enabling such keyword search requires (a) a *preprocessing* step called *Index* that enables databases for keyword search by building the table (FDJT-Tuple-Index) which keeps tuple relationships, and (b) a *Search* step that gets matching results from the published database.

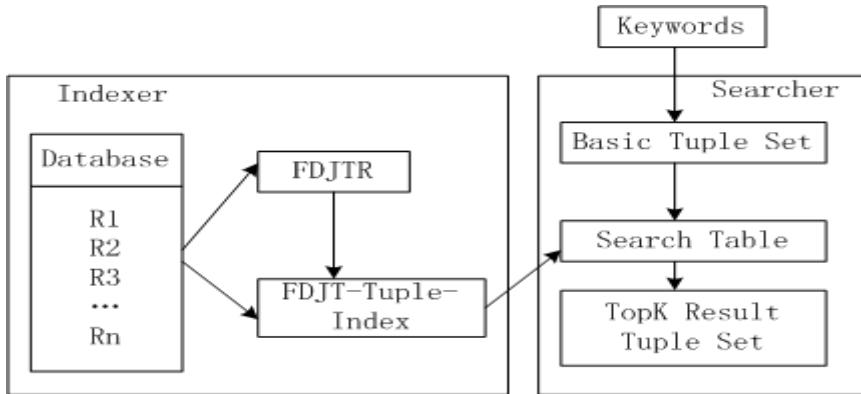


Fig. 2. Architecture of ITREKS

Index step is implemented by model *Indexer* in Figure 2, where Search step is implemented by model *Searcher*.

4.1 Overview of Index and Search Steps

Index: A database is enabled for keyword search through the following steps.

Step 1: A database D is identified, along with its schema graph G.

Step 2: If G is cyclic, turn it into an acyclic schema graph G' with Algorithm 2, which will be discussed in Section 4.2.

Step 3: Given D and G', Indexer generates FDJTR of D using Algorithm 1.

Step 4: FDJT-Tuple-Index table is created for supporting keyword searches, which will be discussed in detail in Section 4.3.

Search: Given a query consisting of a set of keywords, it is answered as follows.

Step 1: For each keyword k, a Basic Tuple Set (BTS) is established by using database full text search functions. Keyword k's BTS is a relation recording all database tuples which have scores to k.

Step 2: Based on BTSs, FDJT-Tuple-Index table and Search Table (see Section 4.4), Searcher finds the results (joinable tuples) which include all keywords. We discuss this step in Section 4.4.

4.2 Acyclization of Database Schema Graph

Given a database schema graph, ITREKS firstly cut off the cycle if the graph is cyclic, so we can use Algorithm 1 to compute the FDJTR of the database.

Figure 3 (a) is schema graph of DBLP database, where , for simplicity, A, W, P and C denote relations Author, Write, Paper and Cite respectively. Figure 3 (b) is a simplest but typical cyclic schema graph. ITREKS revise the cyclic database graph by two operations: cut-off and duplication.

Cut-off: By erasing a less important edge which belongs to the cycle, we can make cyclic schema graph acyclic. Figure 4 shows cut-off revised schema graph in Figure 3, where the schema graph is acyclic but we lost a relation between P and C (in Figure 4 (a)) and relation between B and C (in Figure 4 (b)), which we think is less important. If there isn't a less important relation, we can remove any edge in graph cycle.

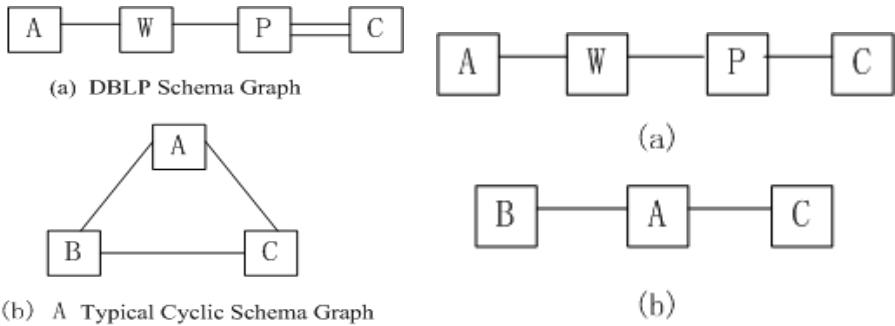


Fig. 3. Two Schema Graph

Fig. 4. Cut-off Revised Schema Graph

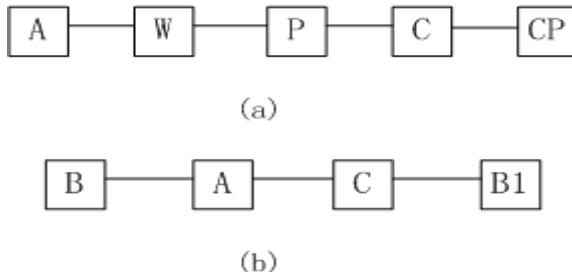


Fig. 5. Duplication Revised Schema Graph

Duplication: By renaming a relation that is in an edge deleted by cut-off operation, we can keep relationship that is deleted by the operation. Figure 5 shows the duplication revised schema graph in Figure 4, where CP is a renamed duplication of relation P (in Figure 4 (a)) and B1 is a renamed duplication of relation B (in Figure 4 (b)).

Pure connective relation: In revised DBLP database graph (see Figure 5 (a)), there are two special relations, W and C, whose attributes are all foreign keys. We call such

relations **pure connective relations**, because the only function of their attributes is to connect tuple and they don't contain indispensable keywords for our keyword search.

In ITREKS, we discard pure connective relations in FDJTR once FDJTR is completely constructed. For example, after computing FDJTR by Algorithm 1 over revised DBLP schema Graph (Figure 5 (a)), the schema of FDJTR is (FDJTid, Aid, Wid, Pid, Cid, CPid). After discard pure connective relations we get FDJTR (FDJTid, Aid, Pid, CPid). For simplicity, we use Aid represent the tuple's id in relation Author. Similarly Pid and PCid are the tuple's id in Paper.

4.3 FDJT-Tuple-Index Table

FDJT-Tuple-Index table index each database tuples with FDJTs. ITREKS builds tuples' relationships by establishing FDJT-Tuple-Index table.

Extended Schema Graph: To build FDJT-Tuple-Index table, ITREKS extends FDJTR as follow:

For each relation in FDJTR, if the relation has edges with other relations in original database schema graph, add these relations and edges to FDJTR. If new added relation is pure connective relation, ITREKES continue add the other relations that have edges with the pure connective relation.

For DBLP database, the **extended schema graph** of FDJTR is shown in Figure 6. Extended Schema reflects the relationship between each database relations and FDJTR. If a relationship is not so important to be indexed, we discard the relative relations in extended schema. For example, in Figure 6, which papers are cited by papers in PC in FDJTR need not to be indexed, ITREKS discard relative relations W and P. Note that extend schema graph is always a tree (is acyclic).

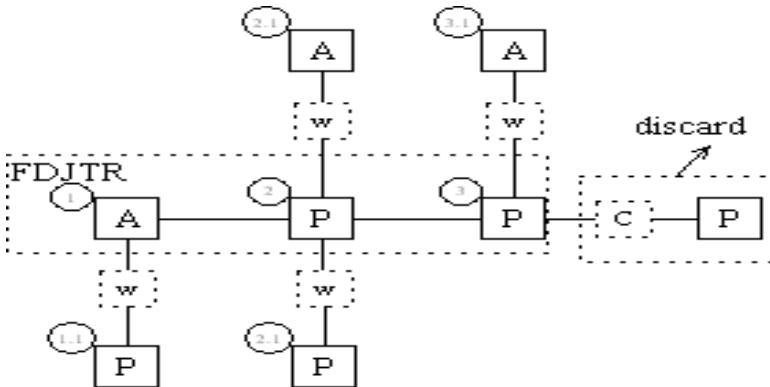


Fig. 6. Extended Schema graph of DBLP

Locator Number: ITREKS gives each relation in extended schema a locator number to records distance and relationship between tuple and FDJT. The number is used when ITREKS calculate the results. ITREKS appoints locator number to relations as follow:

- Let FDJTR has n relations, ITREKS labels each relation in FDJTR with an integer from 1 to n in sequence.
- For other relations in extended schema graph, the locator number consists of two parts divided by a dot. The number in left of the dot (*left number*) is the number of FDJTR's relation connected to it; The number in right of the dot (*right number*) is integer 1.

FDJT-Tuple-Index Table: In ITREKS, FDJT-Tuple-Index table has 4 columns; the first two columns are RN and Tid which identify a database tuple's relation name and rowid. Column FDJTid is rowid of a FDJT in FDJTR that has connection with the tuple. Column N is the locator number representing the relationship between the tuple and the FDJT. The locator number is come from the extended schema graph of the FDJTR. In FDJT-Tuple-Index table, each row records a tuple- FDJT pair and their relationship.

Algorithm 2. Computing FDJT-Tuple-Index Table

Input: database relations R₁,...,R_n, FDJTR of the database

Output: FDJT-Tuple-Index table

1. Extend schema graph of FDJTR of the database to extended schema graph **ESG**.
2. **For each** record R in FDJTR of ESG
3. Insert rowid, relation name and locator number of tuples in R into FDJT-Tuple-Index table.
4. Based on ESG, insert rowid, relation name and locator number of tuples that have relations with R into FDJT-Tuple-Index table.
5. **end for**
6. **return** FDJT-Tuple-Index table

Given a database and its FDJTR, Algorism 2 generates FDJT-Tuple-Index table.

4.4 Searching Step

After FDJT-Tuple-Index table created, ITREKS is ready for keyword search. Given a query consisting of a set of keywords, ITREKS establishes a BTS (Basic Tuple Set) for each keyword k, recording all database tuples which have scores to k. Then based on BTSs, FDJT-Tuple-Index table and Search Table, Searcher finds the results (joinable tuples) which include all keywords.

Definition 5 (BTS). For a keyword k, the **Basic Tuple Set** is a relation $BTS^k = \{t \mid Score(T, k) > 0\}$, which consists of the database tuples with a non-zero score for keyword k.

ITREKS uses Oracle Text Full Text Search Function to build BTSs for each keyword. BTS table consists of 3 columns, RN, Tid and Score, which representing relation name, tuple id and score respectively.

Definition 6 (ST). **Search Table** is a table that is dynamically generated by ITREKS to find joinable tuples at search step. Given keywords k₁,...,k_n, ITREKS generates a ST with $2+k*3$ columns. In ST, a keyword k_i (i=1,...,n) corresponds to 3 columns,

k_i_RN , k_i_Tid and k_i_N , which represent tuples and relationship between the tuples. The other two columns is FDJTid which comes from FDJT-Tuple-Index table and Score of the result.

Definition 7 (Result Tree). **Result Tree** is a tree of joinable tuples based on **extended schema graph** of FDJTR, where each leaf node of the tree contains at least one keyword and the nodes of the tree contain all keywords. The $sizeof(T)$ of a result tree T is the number of edges in T.

Ranking Function: ITREKS uses simple but effective ranking function to rank the result trees for a given query. ITREKS assigns the score of a result tree T in the following way:

$$score(T, Q) = \frac{1}{sizeof(T)} \sum_{i=1}^{sizeof(T)} \sum_{j=1}^k Score(t_i, kw_j)$$

where $Score(t_i, kw_j)$ is the score of a tuple t_i towards keyword kw_j . ITREKS computes $sizeof(T)$ as follow:

Let N be the tuple's locator number that is defined in FDJT-Tuple-Index table.

Let max be the largest left number of N in result tree's leaf nodes; Let min be the smallest left number of N in result tree's leaf nodes. Let r be the sum of the right numbers of result tree's nodes.

Computing the size of T as follow:

$Sizeof(T) = max - min + r$

Given a set of query keywords, ITREKS finds the results by algorithm 3 described below.

Algorithm 3. Generating Results

Input: a query Q, database D, FDJT-Tuple-Index table of the database FDJTTI

Output: Result Trees

1. **For each** keyword ki ($i=1, \dots, n$) in Q **do** {Create BTS_ ki }
2. Sort BTS_ ki in ascending order by the number of the records in the BTS table. We might as well let BTS_ k_1, \dots, k_n be the ascending order list of the BTSs of the keywords
3. Generate search table ST, initially empty
4. Let $F_0 = FDJTTI$
5. $F_1 = F_0$ natural join BTS_ k_1
6. Add relative information(score, FDJT_id, K1_RN, K1_Tid, K1_N) to ST
7. **For** $i=2$ **to** n **do** {
 8. $F_i = F_{i-1}$ natural join BTS_ k_i
 9. Insert relative information(Ki_RN, Ki_Tid, Ki_N) into ST and update relative scores }
10. Remove records that contain null fields from ST
11. Sort records in ST in descending order by their scores
12. For records that have the same values on all fields Ki_RN and Ki_Tid($i=1, \dots, n$) in ST, only keep the record with the highest score and remove others
13. Retrieve the results from ST
14. **Return** result trees

In Algorithm 3, once F_{i-1} natural join BTS_k_i ($i=1,\dots,n$) and put the relative information into ST, ST records relations of joinable tuples based on FDJT and the joined tuples contain all keywords k_j ($j=1,\dots,i$).

5 System Evaluation

Our system is implemented on a PC with Pentium IV 2.8GHz processor and 4GB of RAM, running Windows XP and Oracle 9i. ITREKS has been implemented in Java and connects to the DBMS through JDBC.

We evaluate our tuple relationship indexing and searching system on a 102MB DBLP[12] data set, which we decomposed into 4 relations according to the schema shown in Figure 2. Table 1 summarizes the 4 DBLP relations. The BTS^k toward keyword k is produced by merging the tuples returned by Oracle 9i full-text index on each relation tuple in the database.

Table 2 summarizes FDJTR and FDJT-Tuple-Index (FDJTTI) table which are produced in preprocessing step. Because the FDJTTI table stores the relations between all tuples and FDJTs, it is far larger than other tables. Indexing time includes both FDJT and FDJTTI producing times and only consumes at index step.

Table 1. DBLP dataset characteristics

Relation	#Tuples	Size(MB)
Author	294063	8.62
Write	1000126	36
Paper	446409	51.1
Cite	223013	6.7

Table 2. FDJT and FDJTTI

	FDJT	FDJTTI
Tuples	1248595	39789519
Size(MB)	42.99	2415.92
Time(ms)	155094	3798591

We evaluate search performance by submitting conjunctive keyword queries of length 2, 3, 4 and 5 words. We evaluate returning full ranked result set.

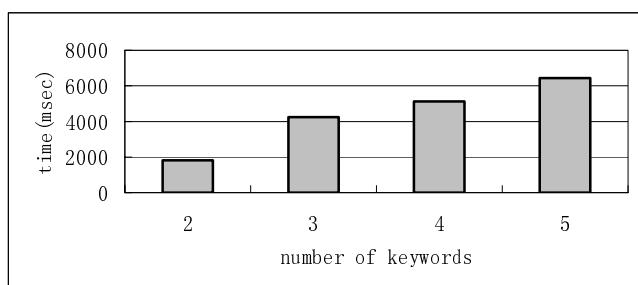


Fig. 7. Query Performance

In each trial, we generate 50 queries by randomly choosing keywords from the keywords set. The reported time for a trial is the average of the 50 query execution times. Figure 7 shows query performance on the DBLP dataset. The performance of queries returning results at less than 7 seconds and alone with the number of keywords increase, the query times do not increase sharply.

6 Conclusion and Future Work

We presented a general architecture for supporting keyword-based search over relational database, and implemented an instantiation of the architecture in our fully implemented system ITREKS. ITREKS indexes tuple relationships in relational database, providing efficient keyword search capabilities over the database. Our system trades online search and offline indexing method to do efficient keyword based search over relational database.

In the future, we will extend our method to semi-structured data like XML and implement our system over more databases.

Acknowledgements

This work is supported by the National Natural Science Foundation of China(No.60473069 and 60496325), and China Grid(No.CNGI-04-15-7A).

References

- [1] Galindo-Legaria, C. Outerjoins as disjunctions. ACM SIGMOD International Conf. on Management of Data, 1994
- [2] Rajaraman, A. and J. D. Ullman. Integrating information by outerjoins and full disjunctions.
- [3] Oracle Text. <http://otn.oracle.com/products/text/index.html>.
- [4] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *Proc. of VLDB*, 2002.
- [5] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. In *Proc. Of VLDB*, 2003.
- [6] Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proc. of ICDE*, 2002.
- [7] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proc. of ICDE*, 2002.
- [8] P. Raghavan. Structured and unstructured search in enterprises. *IEEE Data Engineering Bulletin*, 24(4), 2001.
- [9] S. Dar, G. Entin, S. Geva, , and E. Palmon. Dtl's dataspot: Database exploration using plain languages. In *Proc. Of VLDB*, 1998.
- [10] R. Wheeldon, M. Levene, and K. Keenoy. Search and navigation in relational databases. <http://arxiv.org/abs/cs.DB/0307073>.
- [11] Qi Su, Jennifer Widom. Efficient and Extensible Keyword Search over Relational Databases. Stanford University Technical Report, 2003.
- [12] DBLP bibliography. 2004. <http://www.informatik.uni-trier.de/~ley/db/index.html>

An MBR-Safe Transform for High-Dimensional MBRs in Similar Sequence Matching

Yang-Sae Moon

Department of Computer Science, Kangwon National University
192-1, Hyoja2-Dong, Chunchon, Kangwon 200-701, Korea
ysmoon@kangwon.ac.kr

Abstract. In this paper we propose a formal approach that transforms a high-dimensional MBR itself to a low-dimensional MBR directly, and show that the approach significantly reduces the number of lower-dimensional transformations in similar sequence matching. To achieve this goal, we first formally define a new notion of *MBR-safe*. We say that a transform is MBR-safe if it constructs a low-dimensional MBR by containing all the low-dimensional sequences to which an infinite number of high-dimensional sequences in an MBR are transformed. We then propose an MBR-safe transform based on DFT. For this, we prove the original DFT-based lower-dimensional transformation is not MBR-safe and define a new transform, called *mbrDFT*, by extending definition of DFT. We also formally prove this mbrDFT is MBR-safe. Analytical and experimental results show that our mbrDFT reduces the number of lower-dimensional transformations drastically and improves performance significantly compared with the traditional method.

1 Introduction

Time-series data are the sequences of real numbers representing values at specific points in time. Typical examples of time-series data include stock prices, exchange rates, and weather data [10, 5, 1]. The time-series data stored in a database are called data sequences, and those given by users are called query sequences. Finding data sequences similar to the given query sequence from the database is called *similar sequence matching* [3, 8]. As the distance function $D(X, Y)$ between two sequences $X = \{x_0, x_1, \dots, x_{n-1}\}$ and $Y = \{y_0, y_1, \dots, y_{n-1}\}$ of the same length n , many similar sequence matching models have used L_p -distance ($= \sqrt[p]{\sum_{i=0}^{n-1} |x_i - y_i|^p}$) including the Manhattan distance ($= L_1$), the Euclidean distance ($= L_2$), and the maximum distance ($= L_\infty$) [1, 2, 3, 4, 7, 8, 9].

Most similar sequence matching solutions have used the *lower-dimensional transformation* to store high-dimensional sequences into a multidimensional index [1, 2, 3, 5, 7, 8, 9]. The lower-dimensional transformation has first been introduced in Agrawal et al.'s whole matching solution [1], and widely used in various whole matching solutions [2, 5] and subsequence matching solutions [3, 7, 8, 9]. Recently, it was also used in similar sequence matching on streaming time-series for dimensionality reduction of query sequences or streaming time-series [4]. In

this paper we pay attention to the method of constructing an MBR (Minimum Bounding Rectangle) in similar sequence matching. Previous similar sequence matching solutions use MBRs to reduce the number of points to be stored in a multidimensional index. That is, they do not store individual points directly into the index, but stores only MBRs that contains hundreds or thousands of the low-dimensional points. For example, a low-dimensional MBR in subsequence matching is constructed as follows [3,9]: data sequences are divided into windows; the high-dimensional windows are transformed to low-dimensional points; and an MBR is constructed by containing multiple transformed points. In summary, to construct an MBR to be stored in the index, the existing methods transform tens \sim thousands of high-dimensional sequences (or windows) to low-dimensional sequences (or points) [3,8]. Likewise, the methods should require a huge number of lower-dimensional transformations, and thus in this paper we tackle the problem of how to reduce the number of transformations.

To reduce the number of transformations in constructing low-dimensional MBRs, we propose the lower-dimensional transformation method for high-dimensional MBRs. That is, the method transforms a high-dimensional MBR itself to a low-dimensional MBR directly, where the high-dimensional MBR contains multiple high-dimensional sequences. For this, we first propose a new notion of *MBR-safe*. We say that a transform T is MBR-safe if T satisfies the following property: suppose MBR M is transformed to M^T by T , and sequence X is contained in M , then the transformed sequence X^T by T should also be contained in M^T . If using the notion of MBR-safe, we can construct a low-dimensional MBR by transforming a high-dimensional MBR itself rather than a large number of individual sequences in the MBR. And accordingly, we can reduce the number of transformations required for constructing low-dimensional MBRs.

In this paper we propose an MBR-safe transform based on DFT (Discrete Fourier Transform) [11], which is most widely used as the lower-dimensional transformation. For this, we first prove the original DFT-based lower-dimensional transformation is not MBR-safe. We then define a new transform, called *mbrDFT*, by extending definition of DFT. We also formally prove this mbrDFT is MBR-safe. Through analysis and experiments, we show superiority of the proposed MBR-safe transform. By deriving the computational complexity of constructing a low-dimensional MBR, we analytically show superiority of our mbrDFT. We then empirically show that mbrDFT reduces the number of lower-dimensional transformations drastically and improves performance significantly compared with the traditional method.

2 Related Work

Similar sequence matching can be classified into whole matching and subsequence matching [3]. The *whole matching* [1,2,5] finds data sequences similar to a query sequence, where the lengths of data sequences and the query sequence are all identical. On the other hand, the *subsequence matching* [3,7,8] finds subsequences, contained in data sequences, similar to a query sequence of arbitrary length.

Also, several transform techniques such as moving average transform, shifting & scaling, normalization transform, and time warping have been used in similar sequence matching to solve the problems that the Euclidean distance function has [9]. We note that most similar sequence matching solutions have used the lower-dimensional transformation to use a multidimensional index.

Previous similar sequence matching solutions construct MBRs to reduce the number of points to be stored in the index or to reduce the number of range queries. For example, solutions in [3,9] divide data sequences into windows, transform the windows to low-dimensional points, and finally store MBRs containing multiple transformed points in the index. Similarly, solutions in [7,8] divide a query sequence into windows, transform the windows to low-dimensional points, and finally use MBRs containing multiple transformed points in constructing range queries. Also, recent work for continuous queries on streaming time-series uses the method of constructing MBRs that contain multiple sequences [4]. Likewise, most previous solutions construct MBRs after transforming individual high-dimensional sequences into low-dimensional sequences (points); in contrast, our solution transforms a high-dimensional MBR itself to a low-dimensional MBR directly. Therefore, our solution is quite different from the previous ones in constructing MBRs.

Various transforms including DFT and Wavelet transform are used as the lower-dimensional transformation of high-dimensional sequences. DFT is most widely used in many similar sequence matching solutions [1,3,7,8,9]. Wavelet transform is also used as the lower-dimensional transformation in [2,10]. Besides these transforms, PAA (Piecewise Aggregate Approximation) [5] and SVD (Singular Value Decomposition) [6] were introduced as the lower-dimensional transformation. All these transformations, however, focused on transforming high-dimensional sequences to low-dimensional ones, and they cannot be directly applied to the lower-dimensional transformation of high-dimensional MBRs.

3 Definition of MBR-Safe

We first summarize in Table I the notation to be used throughout the paper. We then formally define the notion of MBR-safe as the following Definition II.

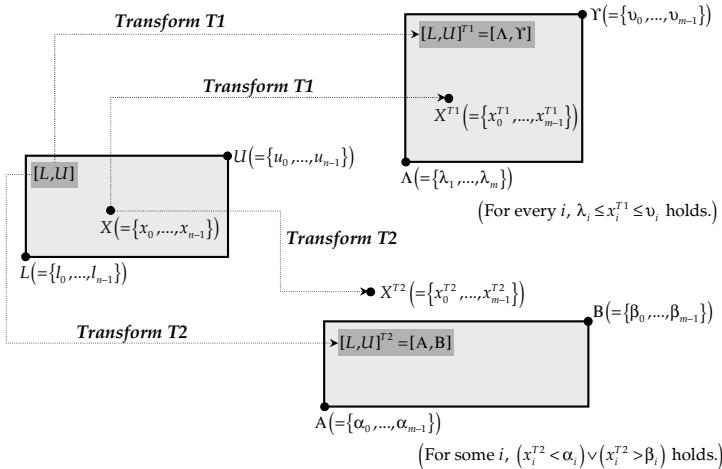
Definition 1. For an n -dimensional sequence X and an n -dimensional MBR $[L, U]$, if a transform T satisfies the following Eq. (II), then we say T is *MBR-safe*.

$$X \in [L, U] \implies X^T \in [L, U]^T \quad (1)$$

Figure II depicts the concept of MBR-safe. In Figure II, transform T_1 is MBR-safe, but T_2 is not. The reason why T_1 is MBR-safe is that, if an arbitrary sequence X is contained in MBR $[L, U]$ (i.e., $X \in [L, U]$), then the transformed sequence X^{T_1} is also contained in the transformed MBR $[L, U]^{T_1}$ (i.e., $X^{T_1} \in [L, U]^{T_1} = [A, Y]$). Analogously, the reason why T_2 is not MBR-safe is that, even though X is contained in $[L, U]$ (i.e., $X \in [L, U]$), X^{T_2} is not contained in $[L, U]^{T_2}$ (i.e., $X^{T_2} \notin [L, U]^{T_2} = [A, B]$).

Table 1. Summary of notation

Symbols	Definitions
X	A high-dimensional sequence. ($= \{x_0, x_1, \dots, x_{n-1}\}$)
X^T	A (low-dimensional) sequence transformed from X by the transform T . ($= \{x_0^T, x_1^T, \dots, x_{m-1}^T\}$)
$[L, U]$	A high-dimensional MBR whose lower-left and upper-right points are L and U , respectively. ($= [\{l_0, l_1, \dots, l_{n-1}\}, \{u_0, u_1, \dots, u_{n-1}\}]$)
$[L, U]^T$ $= [A, T]$	A (low-dimensional) MBR transformed from $[L, U]$ by the transform T . ($= [\{\lambda_0, \lambda_1, \dots, \lambda_{n-1}\}, \{v_0, v_1, \dots, v_{n-1}\}]$)
$X \in [L, U]$	The sequence X is contained in the MBR $[L, U]$. (i.e., for every i , $l_i \leq x_i \leq u_i$)

**Fig. 1.** An MBR-safe transform (T1) and a non-MBR-safe transform (T2)

If using an MBR-safe transform, we can drastically reduce the number of lower-dimensional transformations. In general, previous solutions construct an MBR after tens \sim thousands of lower-dimensional transformations for individual sequences [379]. In contrast, if using the notion of MBR-safe, we can reduce the number of lower-dimensional transformations since we transform the high-dimensional MBR itself to a low-dimensional MBR directly. Figure 2 shows these two methods of constructing a low-dimensional MBR. The upper part of the figure shows an example of using the traditional transform, and the lower part that of using an MBR-safe transform. As shown in the figure, if using the traditional transform, we first transform tens \sim thousands of individual sequences to low-dimensional sequences, and then construct a low-dimensional MBR by containing the transformed sequences. In contrast, if using the MBR-safe transform, we can construct a low-dimensional MBR by simply transforming a high-dimensional

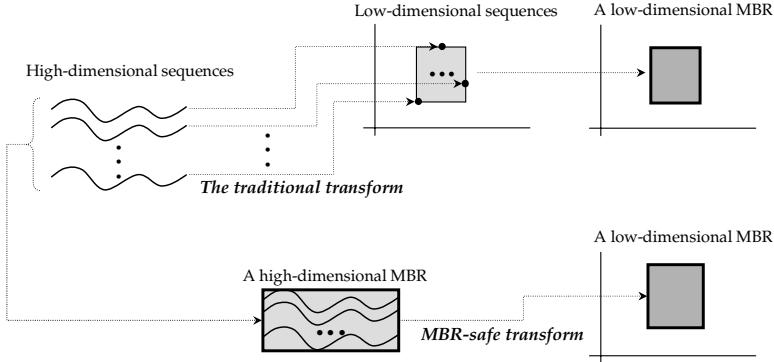


Fig. 2. Two methods of constructing a low-dimensional MBR from high-dimensional sequences

MBR itself rather than a large number of individual sequences. It means that, by using the MBR-safe transform, we can reduce the number of transformations in similar sequence matching.

4 A DFT-Based MBR-Safe Transform

DFT has been most widely used as the lower-dimensional transformation in similar sequence matching [13,7,8,9]. DFT transforms an n -dimensional sequence X to a new n -dimensional sequence $Y (= \{y_0, y_1, \dots, y_{n-1}\})$ in a complex number space, where each complex number y_i is defined as the following Eq. (2) [11]:

$$y_i = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t e^{-j \cdot 2\pi i t / n}, \quad 0 \leq i \leq n - 1. \quad (2)$$

By Euler's formula [11] and definition of complex number, we can rewrite Eq. (2) to Eq. (3) of the real part and imaginary part.

$$y_i = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cos(-2\pi i t / n) + \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \sin(-2\pi i t / n) \cdot j, \quad 0 \leq i \leq n - 1. \quad (3)$$

DFT concentrates most of the energy into the first few coefficients, and thus only a few coefficients extracted from the transformed point Y are used for the lower-dimensional transformation [13]. The following Definition 2 shows the traditional DFT-based lower-dimensional transformation.

Definition 2. The *DFT-based lower-dimensional transformation* transforms an n -dimensional sequence X to a new $m (\ll n)$ -dimensional sequence X^{DFT} of $\{x_0^{DFT}, x_1^{DFT}, \dots, x_{m-1}^{DFT}\}$, where each x_i^{DFT} is obtained by Eq. (4). Also, it transforms an n -dimensional MBR $[L, U]$ to a new m -dimensional MBR $[L, U]^{DFT}$.

whose lower-left and upper-right points are L^{DFT} and U^{DFT} , respectively, i.e., $[L, U]^{DFT} = [L^{DFT}, U^{DFT}]$. In Eq. (4), $\theta = -2\pi[i/2]t/n$ and $0 \leq i \leq m - 1$.

$$x_i^{DFT} = \begin{cases} \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cos \theta, & \text{if } i \text{ is even;} \\ \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \sin \theta, & \text{if } i \text{ is odd.} \end{cases} \quad (4)$$

In similar sequence matching, by using the DFT-based lower-dimensional transformation, we transform a high-dimensional sequence with tens \sim hundreds of dimensions to a low-dimensional sequence with one \sim six dimensions.

The DFT-based lower-dimensional transformation, however, is not MBR-safe. We give in Example 1 a counterexample to show that it is not MBR-safe.

Example 1. Let X be a 4-dimensional sequence of $\{3.00, 2.50, 3.50, 3.00\}$, and $[L, U]$ be a 4-dimensional MBR of $L = \{2.00, 1.00, 3.00, 2.00\}$ and $U = \{4.00, 3.00, 5.00, 4.00\}$. Then, for the given X and $[L, U]$, $X \in [L, U]$ holds. By using the DFT-based lower-dimensional transformation, we now transform the given 4-dimensional sequence and MBR to the 2-dimensional sequence and MBR, respectively. Then, by Definition 2, we can transform X to a new sequence X^{DFT} of $\{6.00, -0.25\}$.¹ Similarly, we can also transform $[L, U]$ to a new MBR $[L, U]^{DFT}$, where $L^{DFT} = \{4.00, -0.50\}$ and $U^{DFT} = \{8.00, -0.50\}$. Here, we note that $-0.50 \leq -0.25 \not\leq -0.50$, that is, $l_2^{DFT} \leq x_2^{DFT} \not\leq u_2^{DFT}$. Thus, for the transformed X^{DFT} and $[L, U]^{DFT}$, $X^{DFT} \in [L, U]^{DFT}$ does not hold. It means that the DFT-based lower-dimensional transformation is not MBR-safe. \square

As noted in Example 1, the DFT-based lower-dimensional transformation is not MBR-safe, and thus we cannot use it for the lower-dimensional transformation of MBRs. Therefore, we introduce a DFT-based MBR-safe transform, called mbrDFT. The following Definition 3 presents a formal definition of mbrDFT.

Definition 3. For an n -dimensional MBR $[L, U]$, mbrDFT is defined as an operation that constructs an $m(\ll n)$ -dimensional MBR $[L, U]^{mbrDFT}$ whose lower-left and upper-right points are Λ and Υ , respectively, in Eq. (5). And, for an n -dimensional sequence X , the mbrDFT-transformed sequence X^{mbrDFT} is identical to X^{DFT} . In Eq. (5), $\theta = -2\pi[i/2]t/n$ and $0 \leq i \leq m - 1$.

$$\lambda_i = \begin{cases} \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} a_t \cos \theta, & \text{if } i \text{ is even;} \\ \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} b_t \sin \theta, & \text{if } i \text{ is odd;} \end{cases}, \quad v_i = \begin{cases} \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} c_t \cos \theta, & \text{if } i \text{ is even;} \\ \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} d_t \sin \theta, & \text{if } i \text{ is odd;} \end{cases},$$

where $\begin{cases} at = lt, ct = ut, & \text{if } \cos \theta \geq 0; \\ at = ut, ct = lt, & \text{if } \cos \theta < 0; \\ bt = lt, dt = ut, & \text{if } \sin \theta \geq 0; \\ bt = ut, dt = lt, & \text{if } \sin \theta < 0. \end{cases}$ (5)

To guarantee MBR-safety of mbrDFT, we intentionally make Λ and Υ in Eq. (5) contain every possible sequence that can be transformed from the original MBR $[L, U]$. The following Theorem 1 shows that mbrDFT is an MBR-safe transform.

¹ In DFT, the imaginary part of the first complex number (i.e., x_1^{DFT}) is always 0. Thus, we use $\{x_0^{DFT}, x_2^{DFT}\}$ instead of $\{x_0^{DFT}, x_1^{DFT}\}$ [38].

Theorem 1. For an n -dimensional sequence X and an n -dimensional MBR $[L, U]$, if $X \in [L, U]$ holds, then $X^{mbrDFT} \in [L, U]^{mbrDFT}$ also holds ($X \in [L, U] \Rightarrow X^{mbrDFT} \in [L, U]^{mbrDFT}$). That is, mbrDFT is MBR-safe.

PROOF: To show $X^{mbrDFT} \in [\Lambda, \Upsilon] (= [L, U]^{mbrDFT})$, we need to prove that $\lambda_i \leq x_i^{mbrDFT} \leq v_i$ holds for every i . We now proceed the proof by two cases: 1) the first case where i of x_i^{mbrDFT} is even, and 2) the second one where i is odd.

1) Assume i is even. Then, $\lambda_i = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} a_t \cos \theta$ and $v_i = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} c_t \cos \theta$. Here, we note that $l_t \leq x_t \leq u_t$ holds for every t ($0 \leq t \leq n-1$) since $X \in [L, U]$ holds by the assumption. Thus, if $\cos \theta$ is positive, $l_t \cos \theta \leq x_t \cos \theta \leq u_t \cos \theta$ holds since $l_t \leq x_t \leq u_t$ holds. Similarly, if $\cos \theta$ is negative, $u_t \cos \theta \leq x_t \cos \theta \leq l_t \cos \theta$ holds. And accordingly, $\sum_{t=0}^{n-1} a_t \cos \theta$, which is obtained by adding $l_t \cos \theta$ if $\cos \theta$ is positive and $u_t \cos \theta$ if $\cos \theta$ is negative, is less than or equal to $\sum_{t=0}^{n-1} x_t \cos \theta$. It means that $\frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} a_t \cos \theta (= \lambda_i)$ is less than or equal to $\frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cos \theta (= x_i^{mbrDFT})$. Analogously, $\sum_{t=0}^{n-1} c_t \cos \theta$, which is obtained by adding $u_t \cos \theta$ if $\cos \theta$ is positive and $l_t \cos \theta$ if $\cos \theta$ is negative, is greater than or equal to $\sum_{t=0}^{n-1} x_t \cos \theta$. Thus, $\frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} c_t \cos \theta (= v_i)$ is greater than or equal to $\frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cos \theta (= x_i^{mbrDFT})$. Therefore, $\lambda_i \leq x_i^{mbrDFT} \leq v_i$ holds for every case where i of x_i^{mbrDFT} is even.

2) Assume i is odd. We can also prove that $\lambda_i \leq x_i^{mbrDFT} \leq v_i$ holds by the similar steps described in the case 1) above.

According to the cases 1) and 2), $\lambda_i \leq x_i^{mbrDFT} \leq v_i$ holds for every i . Therefore, mbrDFT is MBR-safe by Definition 1. \square

The following Example 2 shows that mbrDFT is an MBR-safe transform.

Example 2. As in Example 1, let X be a sequence of $\{3.00, 2.50, 3.50, 3.00\}$, and $[L, U]$ be an MBR of $L = \{2.00, 1.00, 3.00, 2.00\}$ and $U = \{4.00, 3.00, 5.00, 4.00\}$. We now want to transform X and $[L, U]$ using mbrDFT. Then, we can transform X to a new sequence X^{mbrDFT} of $\{6.00, -0.25\}$. Similarly, we can also transform $[L, U]$ to a new MBR $[\Lambda, \Upsilon] (= [L, U]^{mbrDFT})$, where $\Lambda = \{4.00, -1.50\}$ and $\Upsilon = \{8.00, 0.50\}$. Here, we note that both $4.00 \leq 6.00 \leq 8.00$ ($\lambda_0 \leq x_0^{mbrDFT} \leq v_0$) and $-1.50 \leq -0.25 \leq 0.50$ ($\lambda_2 \leq x_2^{mbrDFT} \leq v_2$) hold. Thus, for the mbrDFT-transformed X^{mbrDFT} and $[L, U]^{mbrDFT}$, $X^{mbrDFT} \in [L, U]^{mbrDFT}$ holds. It means that mbrDFT is an MBR-safe transform. \square

The proposed mbrDFT is optimal (i.e., it constructs the smallest MBR) among the DFT-based MBR-safe transforms that convert a high-dimensional MBR itself into a low-dimensional MBR directly. It means that there is no DFT-based MBR-safe transform whose low-dimensional MBR is smaller than that of mbrDFT. We omit the proof of optimality due to space limitation.

5 Computational Complexity Analysis

In this section we analyze computational complexity required to construct a low-dimensional MBR. We analyze two DFT-based transformations: 1) the

traditional method that constructs an MBR after performing the DFT-based lower-dimensional transformation for individual sequences (we simply call this method *orgDFT* and its complexity *orgDFT-complexity*) and 2) the proposed mbrDFT (we call its complexity *mbrDFT-complexity*).

First, orgDFT-complexity depends on the length n and the number m of sequences contained in an MBR. That is, if the computational complexity of a DFT unit operation for a sequence of length n is $O(f(n))$, we can obtain orgDFT-complexity for m sequences as $O(mf(n))$. Here, we know the complexity of one DFT operation for a sequence of length n as $O(n\log n)$ [11]. Thus, orgDFT-complexity for an MBR is to be $O(mn\log n)$. Next, mbrDFT requires only two DFT operations for two sequences, Λ and Υ , respectively. Thus, mbrDFT-complexity for an MBR is to be $O(n\log n)$.

In summary, we derive orgDFT-complexity as $O(mn\log n)$ and mbrDFT-complexity as $O(n\log n)$, respectively. Figure 3(a) shows a graph that presents orgDFT-complexity and mbrDFT-complexity, where we set the length n of sequences to 256 and change the number m of sequences in an MBR from 128 to 1024 by multiples of two. Figure 3(b) shows another graph, where we set m to 256 and change n from 128 to 1024. As shown in the graphs, mbrDFT-complexity is much lower than orgDFT-complexity. Note that Y axes in the graphs have the exponential scale. Also, as m or n increases, the complexity difference between orgDFT and mbrDFT becomes larger. It means that our mbrDFT is very useful and practical for the case where an MBR contains a large number of sequences or the length of sequences is large, i.e., it is suitable for large databases.

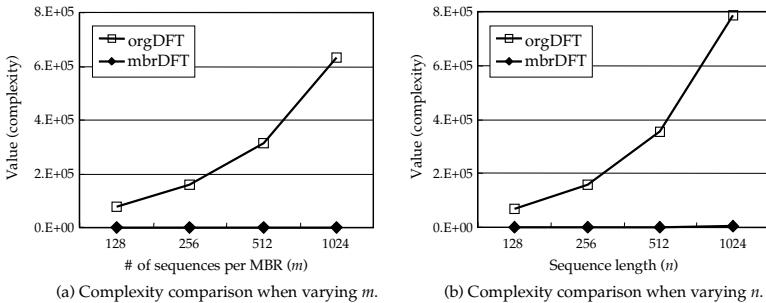


Fig. 3. Comparison of orgDFT-complexity and mbrDFT-complexity

6 Performance Evaluation

6.1 Experimental Data and Environment

We have performed extensive experiments using two types of synthetic data sets. The first data set, used in the previous similar sequence matching works [3, 8, 9], contains a random walk series consisting of one million entries: the first entry is set to 1.5, and subsequent entries are obtained by adding a random value in the range (-0.001, 0.001) to the previous one. We call this data set

WALK-DATA. The second data set contains a synthetic streaming time-series consisting of one million entries: the series is generated using the function $y_i = 100 \cdot [\sin(0.1 \cdot x_i) + 1.0 + i/1000000]$ ($i = 0..999999$) as in [4], where we set x_i to the i -th entry of WALK-DATA. We call this data set *SINE-DATA*.

We generate high-dimensional MBRs by dividing the whole data set into multiple smaller sequences (i.e., sliding windows in [3,8]). In the experiments, we use 128, 256, 512, and 1024 as the length n and the number m of sequences contained in an MBR. As in [4], we transform each high-dimensional sequence, i.e., 128–~1024–dimensional sequence, to a 1–~4–dimensional sequence (point). It means that the number of features extracted by the lower-dimensional transformation is set to one ~ four [1]. As the experimental methods, we compare orgDFT and mbrDFT.

The hardware platform for the experiment is a PC equipped with an Intel Pentium IV 2.80 GHz CPU, 512 MB RAM, and a 70.0GB hard disk. The software platform is GNU/Linux Version 2.6.6 operating system. For the experimental results, we measure the number of transformations and the elapsed time for each method. We also show that, by comparing the boundary-length of the transformed MBRs, the proposed mbrDFT is practically applicable in similar sequence matching.

6.2 Experimental Results

We have performed three experiments. Experiment 1) measures the number of transformations and the elapsed time by varying the number m of sequences in an MBR for the fixed length n of sequences. Experiment 2) performs the same experiment by varying the length n for the fixed number m . Finally, Experiment 3) compares orgDFT and mbrDFT in the boundary-length of MBRs.

Experiment 1) Figure 4 shows the experimental results of orgDFT and mbrDFT. Here, we set the length n of sequences to 256, but change the number m of sequences in an MBR from 128 to 1024 by multiples of two. We set the number of extracted features to two as in [1]. In the experiment, we measure the total number of transformations and the average elapsed time for transforming an MBR. Figure 4(a) shows the numbers of transformations for both WALK-DATA and SINE-DATA; Figures 4(b) and 4(c) show the elapsed times for WALK-DATA and SINE-DATA, respectively. As shown in Figure 4(a), our mbrDFT drastically reduces the number of transformations over orgDFT. It is because orgDFT has to consider all the individual sequences in an MBR; in contrast, mbrDFT requires only two transformations for an MBR. Figures 4(b) and 4(c) show that mbrDFT also reduces the elapsed time significantly over orgDFT. As we analyzed in Figure 3(a) in Section 5, the more number of sequences in an MBR causes the more performance difference between orgDFT and mbrDFT. In summary, mbrDFT drastically reduces the number of transformations to $\frac{1}{136}$ of that for orgDFT on the average, and also significantly improves performance by 31 times that for orgDFT on the average.

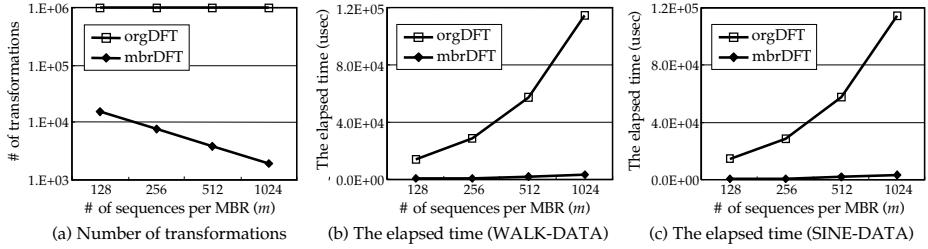


Fig. 4. Experimental results when varying the number m of sequences in an MBR

Experiment 2) Figure 5 shows the results when we set the number m of sequences in an MBR to 256, but change the length n of sequences from 128 to 1024 by multiples of two. As in Experiment 1), we measure the total number of transformations and the average elapsed time for transforming an MBR. From Figure 5(a), we note that the numbers of transformations are not changed even as the length of sequences increases. It is because the numbers are dependent on the number of sequences in orgDFT or the number of MBRs in mbrDFT, but are not dependent on the length of sequences in both orgDFT and mbrDFT. As shown in Figures 5(b) and 5(c), mbrDFT significantly reduces the elapsed time over orgDFT. In particular, as we analyzed in Figure 3(b) in Section 5, the larger length of sequences causes the more performance difference between orgDFT and mbrDFT.

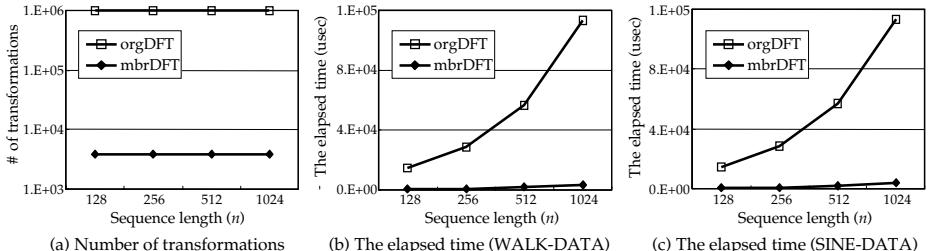


Fig. 5. Experimental results when varying the length n of sequences

Experiment 3) In this experiment, we compare the methods in the average boundary-length of MBRs. Here, the *boundary-length of an MBR* is defined as the sum of the length of each dimension in the MBR, i.e., the boundary-length of $[L, U]$ is defined as $\sum_{i=0}^{n-1} (u_i - l_i)$. Figure 6 compares orgDFT and mbrDFT in the average boundary-length of MBRs. Here, we set both the length n and the number m of sequences in an MBR to 256, but increment the number of extracted dimensions (features) from one to four. As shown in the figure, the average boundary-length in mbrDFT is longer than that in orgDFT if the number of extracted dimensions is greater than two. It is because our mbrDFT considers

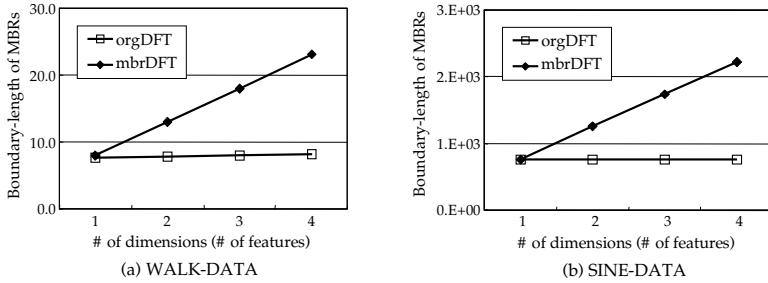


Fig. 6. Comparison of orgDFT and mbrDFT in the average boundary-length of MBRs

an infinite number of every possible sequence that can be contained in a high-dimensional MBR, while orgDFT does a finite number of real sequences in the MBR. On the other hand, if the number of extracted dimensions is one, there is only a little difference (0.2%~2.6%) in the boundary-length. As experimented in [1], DFT concentrates most of energy into the first dimension, and thus we can say that our mbrDFT is much more useful if we extract only one or two dimensions.

7 Conclusions

In this paper we have proposed a formal approach that transforms a high-dimensional MBR itself to a low-dimensional MBR directly. We have noted that most similar sequence matching solutions required a huge number of lower-dimensional transformations to construct low-dimensional MBRs to be stored in the index. To solve this problem, we have introduced a new notion of MBR-safe and proposed MBR-safe transforms that can reduce the number of lower-dimensional transformations drastically.

We can summarize our work as the following three contributions. First, we formally defined the notion of MBR-safe. If using the notion of MBR-safe, we can construct a low-dimensional MBR by transforming a high-dimensional MBR itself rather than a large number of individual sequences. Second, we proposed a DFT-based MBR-safe transform. For this, we first proved the traditional DFT-based lower-dimensional transformation is not MBR-safe. We then introduced a new transform, called mbrDFT, and formally proved in Theorem 1 it is MBR-safe. Third, through analysis and experiments, we showed superiority of our MBR-safe transform.

These results indicate that our MBR-safe transforms will provide a useful framework for a variety of applications that require the lower-dimensional transformation of high-dimensional MBRs. Therefore, as the further research, we will try to apply the MBR-safe transform to real applications such as similarity search, multimedia data retrieval, and GIS.

Acknowledgements

This work was supported by the Ministry of Science and Technology (MOST)/Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

References

1. Agrawal, R., Faloutsos, C., and Swami, A., "Efficient Similarity Search in Sequence Databases," In *Proc. the 4th Int'l Conf. on Foundations of Data Organization and Algorithms*, pp. 69-84, Oct. 1993.
2. Chan, K.-P., Fu, A. W.-C., and Yu, C. T., "Haar Wavelets for Efficient Similarity Search of Time-Series: With and Without Time Warping," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 15, No. 3, pp. 686-705, Jan./Feb. 2003.
3. Faloutsos, C., Ranganathan, M., and Manolopoulos, Y., "Fast Subsequence Matching in Time-Series Databases," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 419-429, May 1994.
4. Gao, L. and Wang, X. S., "Continually Evaluating Similarity-based Pattern Queries on a Streaming Time Series," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 370-381, June 2002.
5. Keogh, E. J., Chakrabarti, K., Mehrotra, S., and Pazzani, M. J., "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases," In *Proc. of Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 151-162, May 2001.
6. Korn, F., Jagadish, H. V., and Faloutsos, C., "Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences," In *Proc. of Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 289-300, June 1997.
7. Lim, S.-H., Park, H.-J., and Kim, S.-W., "Using Multiple Indexes for Efficient Subsequence Matching in Time-Series Databases," In *Proc. of the 11th Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, pp. 65-79, Apr. 2006.
8. Moon, Y.-S., Whang, K.-Y., and Han, W.-S., "General Match: A Subsequence Matching Method in Time-Series Databases Based on Generalized Windows," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 382-393, June 2002.
9. Moon, Y.-S. and Kim, J., "A Single Index Approach for Time-Series Subsequence Matching that Supports Moving Average Transform of Arbitrary Order," In *Proc. of the 10th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD)*, pp. 739-749, Apr. 2006.
10. Natsev, A., Rastogi, R., and Shim, K., "WALRUS: A Similarity Retrieval Algorithm for Image Databases," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 16, No. 3, pp. 301-316 , Mar. 2004.
11. Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, 2nd Ed., 1992.

Mining Closed Frequent Free Trees in Graph Databases

Peixiang Zhao and Jeffrey Xu Yu

The Chinese University of Hong Kong, China
`{pxzhao, yu}@se.cuhk.edu.hk`

Abstract. Free tree, as a special graph which is connected, undirected and acyclic, has been extensively used in bioinformatics, pattern recognition, computer networks, XML databases, etc. Recent research on structural pattern mining has focused on an important problem of discovering frequent free trees in large graph databases. However, it can be prohibitive due to the presence of an exponential number of frequent free trees in the graph database. In this paper, we propose a computationally efficient algorithm that discovers only *closed frequent free trees* in a database of labeled graphs. A free tree t is *closed* if there exist no supertrees of t that has the same frequency of t . Two pruning algorithms, the *safe position pruning* and the *safe label pruning*, are proposed to efficiently detect unsatisfactory search spaces with no closed frequent free trees generated. Based on the special characteristics of free tree, the *automorphism-based pruning* and the *canonical mapping-based pruning* are introduced to facilitate the mining process. Our performance study shows that our algorithm not only reduces the number of false positives generated but also improves the mining efficiency, especially in the presence of large frequent free tree patterns in the graph database.

1 Introduction

Recent research on frequent pattern discovery has progressed from mining itemsets and sequences to mining structural patterns including (ordered, unordered, free) trees, lattices, graphs and other complicated structures. Among all these structural patterns, graph, a general data structure representing relations among entities, has been widely used in a broad range of areas, such as bioinformatics, chemistry, pattern recognition, computer networks, etc. In recent years, we have witnessed a number of algorithms addressing the frequent graph mining problem [5946]. However, discovering frequent graph patterns comes with expensive cost. Two computationally expensive operations are unavoidable: (1) to check if a graph contains another graph (in order to determine the frequency of a graph pattern) is an instance of *subgraph isomorphism* problem, which is NP-complete [3]; and (2) to check if two graphs are isomorphic (in order to avoid creating a candidate graph for multiple times) is an instance of *graph isomorphism* problem, which is not known to be either P or NP-complete [3].

With the advent of XML and the need for mining semi-structured data, a particularly useful family of general graph — free tree, has been studied and

applied extensively in various areas such as bioinformatics, chemistry, computer vision, networks, etc. Free tree — the connected, undirected and acyclic graph, is a generalization of linear sequential patterns, and hence reserves plenty of structural information of databases. At the same time, it is a specialization of general graph, therefore avoids undesirable theoretical properties and algorithmic complexities incurred by graph. As the middle ground between two extremes, free tree has provided us a good compromise in data mining research [82].

Similar to frequent graph mining, the discovery of frequent free trees in a graph database shares a common *combinatorial explosion* problem: the number of frequent free trees grows exponentially although most free trees deliver nothing interesting but redundant information if all of them share the same frequency. This is the case especially when graphs of a database are strongly correlated.

Our work is inspired by mining *closed frequent itemsets and sequences* in [7]. According to [7][11], a frequent pattern \mathcal{I} is *closed* if there exists no proper super-pattern of \mathcal{I} with the same frequency in the dataset. In comparison to frequent free trees, the number of closed ones is dramatically small. At the same time, closed frequent free trees maintain the same information (w.r.t frequency) as that held by frequent free trees with less redundancy and better efficiency.

There are several previous studies on discovering closed frequent patterns among large tree or graph databases. **CMTreeMiner** [4] discovers all closed frequent ordered or unordered trees in a rooted-tree database by traversing an *enumeration tree*, a special data structure to enumerate all frequent (ordered or unordered) subtrees in the database. However, some elegant properties of ordered (unordered) trees do not hold in free trees, which makes infeasible to apply their pruning techniques directly to mine closed frequent free trees. **CloseGraph** [10] discovers all closed frequent subgraphs in a graph database by traversing a search space representing the complete set of frequent subgraphs. The novel concepts of *equivalent occurrence* and *early termination* help CloseGraph prune certain branches of the search space which produce no closed frequent subgraphs. We can directly use CloseGraph to mine closed frequent free trees because free tree is a special case of general graph, but CloseGraph will introduce a lot of inefficiencies. First, all free trees are computed as general graphs while the intrinsic characteristics of free tree are omitted; Second, the early termination may fail and CloseGraph may miss some closed frequent patterns. Although this failure of early termination can be detected, the detection operations should be applied case-by-case, which introduce a lot of complexities.

In this paper, we fully study the closed frequent free tree mining problem and develop an efficient algorithm, **CFFTree** which is short for **Closed Frequent, Free Tree** mining, to systematically discover the complete set of closed frequent free trees in large graph databases. The main contributions of this paper are: (1) We first introduce the concept of closed frequent free trees and study its properties and its relationship to frequent free trees; (2) Our algorithm **CFFTree** depth-first traverses the enumeration tree to discover closed frequent free trees. Two original pruning algorithms, the *safe position pruning* and the *safe label pruning* are proposed to prune search branches of the enumeration tree in the early

stage, which is confirmed to output no desired patterns; (3) Based on the intrinsic characteristics of free tree, we propose the *automorphism-based pruning* and the *canonical mapping-based pruning* to alleviate the expensive computation of equivalent occurrence sets and candidate answer sets during the mining process. We carried out different experiments on both synthetic data and real application data. Our performance study shows that **CFFTTree** outperforms up-to-date frequent free mining algorithms by a factor of roughly 10. To the best of our knowledge, **CFFTTree** is the first algorithm that, instead of using post-processing methods, directly mines closed frequent free trees from graph databases.

The rest of the paper is organized as follows. Section 2 provides necessary background and detailed problem statement. We study the closed frequent free tree mining problem in Section 3, and propose a basic algorithmic framework to solve the problem. Advanced pruning algorithms are presented in Section 4. Section 5 formulates our algorithm, **CFFTTree**. In Section 6, we report our performance study and finally, we offer conclusions in Section 7.

2 Preliminaries

A *labeled graph* is defined as a 4-tuple $G = (V, E, \Sigma, \lambda)$ where V is a set of vertices, E is a set of edges (unordered pairs of vertices), Σ is a set of labels, and λ is a labeling function, $\lambda : V \cup E \rightarrow \Sigma$, that assigns labels to vertices and edges. A *free tree*, denoted *ftree*, is a special undirected labeled graph that is connected and acyclic. Below, we call a *ftee* with n vertices a n -*ftee*.

Let t and s be two *ftees*, and g be a graph. t is a *subtree* of s (or s is the *supertree* of t), denoted $t \subseteq s$, if t can be obtained from s by repeatedly removing vertices with degree 1, a.k.a *leaves* of the tree. Similarly, t is a *subtree* of a graph g , denoted $t \subseteq g$, if t can be obtained by repeatedly removing vertices and edges from g . *Ftees* t and s are *isomorphic* to each other if there is a one-to-one mapping from the vertices of t to the vertices of s that preserves vertex labels, edge labels, and adjacency. An *automorphism* is an isomorphism that maps from a *ftee* to itself. A *subtree isomorphism* from t to g is an isomorphism from t to some subtree(s) of g .

Given a graph database $\mathcal{D} = \{g_1, g_2, \dots, g_N\}$ where g_i is a graph ($1 \leq i \leq N$). The problem of *ftee mining* is to discover the set of all frequent *ftees*, denoted FS , where $t \in FS$ iff the ratio of graphs in \mathcal{D} that has t as its subtree is greater than or equal to a user-given threshold ϕ . Formally, let t be a *ftee* and g_i be a graph. We define

$$\varsigma(t, g_i) = \begin{cases} 1 & \text{if } t \subseteq g_i \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

and

$$\sigma(t, \mathcal{D}) = \sum_{g_i \in \mathcal{D}} \varsigma(t, g_i) \quad (2)$$

where $\sigma(t, \mathcal{D})$ denotes the *frequency* or *support* of t in \mathcal{D} . The frequent *ftee* mining problem is to discover the *ftee* set FS of \mathcal{D} which satisfies

$$FS = \{t \mid \sigma(t, \mathcal{D}) \geq \phi N\} \quad (3)$$

The problem of *closed frequent ftree mining* is to discover the set of frequent *frees*, denoted CFS , where $t \in CFS$ iff t is frequent and the support of t is strictly larger than that of any supertree of t . Formally, the closed frequent *ftee* mining problem is to discover the *ftee* set CFS of \mathcal{D} which satisfies

$$CFS = \{t \mid t \in FS \wedge \forall t' \supset t, \sigma(t, \mathcal{D}) > \sigma(t', \mathcal{D})\} \quad (4)$$

Since CFS contains no *ftee* that has a supertree with the same support, we have $CFS \subseteq FS$.

3 Closed Frequent Ftree Mining: Proposed Solutions

Based on the definition in Eq.(4), a naive two-step algorithm of discovering CFS from \mathcal{D} can be easily drafted. First, using current frequent *ftee* mining algorithms to discover FS from \mathcal{D} ; Second, for each $t \in FS$, examining all $t' \in FS$ where $t \subset t'$ to tell whether t' satisfies $\sigma(t', \mathcal{D}) < \sigma(t, \mathcal{D})$. This algorithm is straightforward, but far from efficient. It indirectly discovers CFS by computing FS in the first place whose size is exponentially larger than that of CFS . The postprocessing operation of filtering non-closed frequent *frees* from FS also incurs unnecessary computation. We want an alternative method which directly computes CFS instead of computing FS in advance, i.e., under the traditional search space for mining frequent *frees*, efficient pruning algorithms should be proposed to detect branches that do not correspond to closed frequent *frees* as early as possible, and prune them to avoid unnecessary computation, which finally facilitate the total mining process.

In [12], we demonstrate F3TM, a fast frequent *ftee* mining algorithm, which outperforms up-to-date algorithms FreeTreeMiner[28] by an order of magnitude. In F3TM, an enumeration tree representing the search space of all frequent *frees* is built by a pattern-growth approach. Given a frequent n -*ftee* t , the potential frequent $(n + 1)$ -*ftee* t' originated from t is generated as

$$t' = t \circ_{ef} v, v \in \Sigma \quad (5)$$

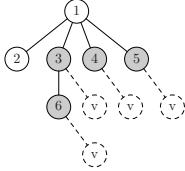
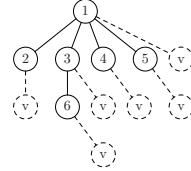
where ef means pattern growth can be conducted on the *extension frontier* of t instead of each vertex of t , while at the same time ensuring the completeness of frequent *frees* discovered from the graph database. Figure ④ illustrates the extension frontier of a *ftee*, which is composed of vertices 3, 4, 5 and 6, and the candidate generation of t , based on Eq. ⑤.

For each frequent *ftee* in the enumeration tree discovered by F3TM, we can check the closeness condition in Eq. ④. Given a frequent n -*ftee* t , its *immediate supertree set*, denoted $CS(t)$, which contains all $(n + 1)$ -*frees* $t' \supset t$ can be generated as

$$CS(t) = \{t' \mid t' = t \circ_x v, v \in \Sigma\} \quad (6)$$

where x means v can be grown on any vertex of t , which is shown in Figure ②. t 's *immediate frequent supertree set*, denoted $FS(t)$, which contains all frequent $(n + 1)$ -*frees* $t' \supset t$ can be generated as

$$FS(t) = \{t' \mid t' \in CS(t) \wedge \sigma(t, \mathcal{D}) \geq \phi N\} \quad (7)$$

**Fig. 1.** $t' = t \circ_{ef} v$ **Fig. 2.** $t' = t \circ_x v$

Given a frequent *f*tree $t' \in FS(t)$, we denote the vertex which is grown on t to get t' as $(t' - t)$, and the vertex of t at which $(t' - t)$ is grown on as p , i.e., the parent of $(t' - t)$ in t' .

The basic algorithmic framework for mining closed frequent *f*trees can be formalized as follows: if for every $t' \in FS(t)$, $\sigma(t', \mathcal{D})$ is strictly smaller than $\sigma(t, \mathcal{D})$, then t is closed; Otherwise, t is non-closed, i.e., we can tell the closeness of t by checking the support values of all its immediate frequent supertrees in $FS(t)$ during the traversal of the enumeration tree for mining frequent *f*trees.

4 Pruning the Search Space

In the previous section, we traverse the enumeration tree to discover all frequent *f*trees in a graph database. However, the final goal of our algorithm is to find only closed frequent *f*trees. Therefore, it is not necessary to grow the complete enumeration tree, because under certain conditions, some branches of the enumeration tree are guaranteed to produce no closed frequent *f*trees and therefore can be pruned efficiently. In this section, we introduce algorithms that prune unwanted branches of the search space.

4.1 Equivalent Occurrence

Given a *f*tree t and a graph $g \in \mathcal{D}$, let $f(t, g)$ represents a subtree isomorphism from t to g . $f(t, g)$ is also referred to as an *occurrence* of t in g . Notice that t can occurs more than once in g . Let $\omega(t, g)$ denote the number of occurrences of t in g . The number of occurrences of t in a graph database \mathcal{D} can be formally defined as

Definition 1. Given a *f*tree t and a graph database $\mathcal{D} = \{g_1, g_2, \dots, g_N\}$, the number of occurrence of t in \mathcal{D} is the sum of the number of subtree isomorphisms of t in $g_i \in \mathcal{D}$, i.e., $\sum_{i=1}^N \omega(t, g_i)$, denoted by $\mathcal{O}(t, \mathcal{D})$.

Suppose a *f*tree $t' = t \circ_x v$, f is a subtree isomorphism of t in g and f' is a subtree isomorphism of t' in g . If $\exists \rho$, ρ is subtree isomorphism of t in t' , i.e., $\forall v, f(v) = f'(\rho(v))$, we call t and t' *simultaneously occur* in graph g . Intuitively, as we can derive t' from t by $t' = t \circ_x v$, we can get t' in the same pattern-growth way from t in g . We denote the number of such *simultaneous occurrences* of t' w.r.t t in g by $\omega(t, t', g)$. Similarly, the number of simultaneous occurrences of t' w.r.t t in \mathcal{D} is defined as

Definition 2. Given a *f*tree $t' = t \circ_x v$ and a graph database $\mathcal{D} = \{g_1, g_2, \dots, g_N\}$, the number of simultaneous occurrence of t' w.r.t. t in \mathcal{D} is the sum of the number

of simultaneous occurrences of t' w.r.t t in $g_i \in \mathcal{D}$, i.e., $\sum_{i=1}^N \omega(t, t', g_i)$, denoted by $\mathcal{SO}(t, t', D)$.

Definition 3. Given $t' = t \circ_x v$ and a graph database $\mathcal{D} = \{g_1, g_2, \dots, g_N\}$, if $\mathcal{O}(t, D) = \mathcal{SO}(t, t', D)$, we say that t and t' have **equivalent occurrences**.

Lemma 1. For a frequent ftree t in the enumeration tree, if there exists a $t' \in FS(t)$ such that (1) t and t' have equivalent occurrences; (2) the vertex $(t' - t)$ is not grown on the extension frontier of any descendants of t , including t , in the enumeration tree, then (1) t is not a closed frequent ftree and (2) for each child t'' of t in the enumeration tree, there exists at least one supertree t''' of t'' , such that t''' and t'' have equivalent occurrences.

Proof. The first statement can be easily proved. Since t and t' have equivalent occurrences in \mathcal{D} , then $\mathcal{O}(t', D) = \mathcal{O}(t, D)$. For the second statement, we notice that $(t' - t)$ occurs at each occurrence of t in \mathcal{D} , so it occurs at each occurrence of t'' in \mathcal{D} . In addition, the vertex $(t' - t)$ never be grown on the extension frontier of any descendant of t , so it will not be a vertex of t'' (Notice t'' is a child of t in the enumeration tree by growing a vertex on t 's extension frontier). Therefore, we can obtain t''' by adding $(t' - t)$ on t'' , so that t'' and t''' have equivalent occurrences.

By inductively applying Lemma 1 to t and all t 's descendants in the enumeration tree, we can conclude that all branches originated from t in the enumeration tree are guaranteed to produce no closed frequent ftrees. However, the conditions mentioned in Lemma 1, especially the condition (2) is hard to be justified. Since when mining frequent ftree t , we have no information of all t 's descendants in the enumeration tree. The following sections will present more detailed techniques to prune the search space.

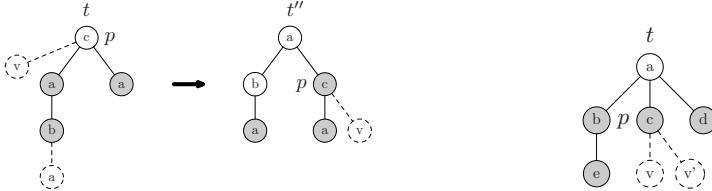
4.2 The Safe Position Pruning

Given a ftree t and a vertex $v \in t$, the *depth* of v can be defined as follows

$$\text{depth}(v) = \begin{cases} 1 & \text{if } v \text{ is a leaf} \\ \min_{u \in t, u \text{ is child of } v} \{\text{depth}(u) + 1\} & \text{otherwise} \end{cases} \quad (8)$$

Intuitively, the depth of a vertex v is the minimum number of vertices from v to the nearest leaf of t . For a frequent ftree $t' \in FS(t)$ where t and t' have equivalent occurrences, the vertex $(t' - t)$ can be grown at different positions, i.e., there are the following possibilities for the position of p in t . (1) $\text{depth}(p) \leq 2$ and p is on the extension frontier of t ; (2) $\text{depth}(p) \leq 2$ but p is not on the extension frontier; (3) $\text{depth}(p) > 2$.

If p occurs in position (1), vertex $(t' - t)$ is grown on the extension frontier of t . If p occurs in position (2), there are possibilities that for some descendant t'' of t in the enumeration tree, the vertex p can still be on the extension frontier of t'' . A example is shown in Figure 3. In frequent ftree t , $\text{depth}(p) = 2$ and p is not located on the extension frontier. After the vertex a is grown on the extension frontier (vertex b), we get another frequent ftree t'' in which p is now located on

**Fig. 3.** A Special Case in Position (2)**Fig. 4.** The Safe Label Pruning

the extension frontier. So the first two possible positions of p are *unsafe* when growing vertex $(t' - t)$, which disallows the conditions mentioned in Lemma \square .

The following theorem shows that only position (3) of p is *safe* to grow the vertex $(t' - t)$, while not violating the conditions mentioned in Lemma \square .

Theorem 1. *For a frequent ftree $t' \in FS(t)$ such that t and t' have equivalent occurrences in \mathcal{D} . If $\text{depth}(p) > 2$, then neither t nor any t 's descendants in the enumeration tree can be closed.*

Proof. Since for every vertex u on the extension frontier of a ftree, it is located at the bottom two levels, i.e., $\text{depth}(u) \leq 2$. If $\text{depth}(p) > 2$, the vertex p can never appear on the extension frontier of any ftree, i.e., the vertex $(t' - t)$ will not be grown on the extension frontier of any descendant of t , including t , in the enumeration tree. According to Lemma \square , the branches originated from t can not generate closed frequent ftrees.

The pruning algorithm mentioned in Theorem \square is called the *safe position pruning*, since the vertex $(t' - t)$ can only be grown on a safe vertex $p \in t$, where $\text{depth}(p) > 2$. Given a n -ftree, the depth of every vertex of t can be computed in $O(n)$, so the safe position pruning is quite efficient to testify whether a certain branch in the enumeration tree should be pruned or not.

4.3 The Safe Label Pruning

If p is on the extension frontier of t , obviously, $\text{depth}(p) \leq 2$. We can not prune t from the enumeration tree. However, depending on the vertex label of $(t' - t)$, we can still possibly prune some children of t in the enumeration tree.

Theorem 2. *For a frequent ftree $t' \in FS(t)$ such that t and t' have equivalent occurrences in \mathcal{D} , if p is located on the extension frontier of t , we do not need to grow t by adding to p a new vertex with label lexicographically greater than $(t' - t)$.*

Proof. For any $t'' \in FS(t)$ such that p is the parent of $(t'' - t)$ and $(t'' - t)$ is lexicographically greater than $(t' - t)$, a ftree $t''' = t'' \circ_p (t' - t)$ have equivalent occurrence with t'' and $t''' \in FS(t'')$. Note $t'' \circ_p (t' - t)$ means growing vertex $(t' - t)$ on p of ftree t'' . According to Lemma \square , t'' is not closed. And for every descendant of t'' in the enumeration tree, $(t' - t)$ never be grown on its extension frontier. Because during frequent ftrees mining, we generate candidates in a lexicographical order. Since $(t'' - t)$ is lexicographically greater than $(t' - t)$,

the vertex $(t' - t)$ will not be reconsidered to be grown on t'' and all t'' 's descendants in the enumeration tree. According to Lemma 11, neither t'' nor any of its descendants can be closed.

The pruning algorithm mentioned in Theorem 2 is called the *safe label pruning*. The vertex label of $(t' - t)$ is *safe* because all vertices with labels lexicographically greater than $(t' - t)$ can be exempted from growing on p of t , and all descendants of corresponding *freeses* in the enumeration tree are also pruned. An example is shown in Figure 4. p is located on the extension frontier of t and $v = (t' - t)$. If v'' 's label is lexicographically greater than v 's label, the frequent *freeses* $t'' = t \circ_p v'$ and the frequent *freeses* $t''' = t'' \circ_p v$ have equivalent occurrences, so that t'' is not closed. Similarly, all t'' 's descendants in the enumeration tree are not closed, either.

4.4 Efficient Computation of $FS(t)$

Based on the above analysis, both candidate generation and closeness test of the frequent *freeses*, t , need to compute $FS(t)$. Depending on if t can be pruned from the enumeration tree during closed frequent *freeses* mining, we can divide $FS(t)$ into the following mutually exclusive subsets:

$$\begin{aligned} EO(t) &= \{t' \in FS(t) \mid t' \text{ and } t \text{ have equivalent occurrences}\} \\ EN(t) &= \{t' \in FS(t) \mid \sigma(t, \mathcal{D}) = \sigma(t', \mathcal{D})\} \\ F(t) &= \{t' \in FS(t) \mid t' \text{ is frequent}\} \end{aligned}$$

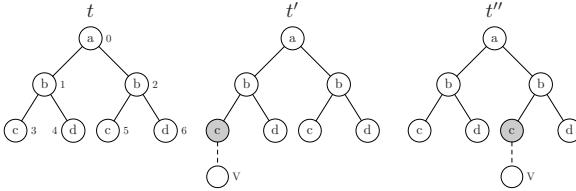
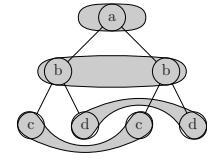
Based on Theorem 11 and Theorem 2, the set $EO(t)$ can be further divided into the following mutually exclusive subsets:

$$\begin{aligned} EO_1(t) &= \{t' \in EO(t) \mid p \in t \text{ is safe}\} \\ EO_2(t) &= \{t' \in EO(t) \mid p \text{ is on the extension frontier of } t\} \\ EO_3(t) &= EO(t) - EO_1(t) - EO_2(t) \end{aligned}$$

When computing the sets mentioned above, we map t to each occurrence in $g_i \in \mathcal{D}$ and select the possible vertex $(t' - t)$ to grow. However, this procedure is far from efficient since a lot of redundant t' are generated. Now we study how to speed up the computation of $FS(t)$ based on the characteristics of *freeses*. The detailed analysis can be found in [2].

Automorphism-based Pruning: In the example shown in Figure 5. The leftmost *freeses* t is a frequent 7-*freeses*, where vertices are identified with a unique number as *vertex id*. When growing a new vertex v on vertex 3 of t , we get a 8-*freeses* $t' \in CS(t)$, shown in the middle of Figure 5. However, when growing v on vertex 5 of t , we get another 8-*freeses* $t'' \in CS(t)$, shown on the right of Figure 5. Notice $t' = t''$ in the sense of *freeses* isomorphism, so t'' can be pruned when computing $FS(t)$.

Based on the observation mentioned above, We propose an *automorphism-based pruning* algorithm to efficiently avoid redundant generation of *freeses* in $FS(t)$. Given a *freeses*, all vertices can be partitioned into different equivalence classes based on *freeses* automorphism. Figure 6 shows how to partition vertices

**Fig. 5.** $t', t'' \in CS(t)$ and $t' = t''$ **Fig. 6.** Equivalence Class

of t in Figure 5 into four equivalence classes. When computing $FS(t)$, only one representative for each equivalence class of t is considered, instead of growing vertices on every position within an equivalence class.

Canonical Mapping-based Pruning: When computing $FS(t)$, we maintain *mappings* from t to all its occurrences in $g_i \in \mathcal{D}$. However, there exist redundant mappings because of *f*tree automorphism. Given a n -*f*tree t , and assume that the number of equivalence classes of t is c , and the number of vertices in each equivalence class C_i is n_i , for $1 \leq i \leq c$. The number of mappings from t to an occurrence in g_i is computed as $\omega(t, g_i) = \prod_{i=1}^c (n_i)!$. When either the number of equivalence classes, or the number of vertices in some equivalence class is large, $\omega(t, g_i)$ can be huge. However, among all mappings describing the same occurrence of $t \in g_i$, one out of $\prod_{i=1}^c (n_i)!$ mappings is selected as *canonical mapping* and all computation of $FS(t)$ is based on the canonical mapping of t in \mathcal{D} . While other $(\prod_{i=1}^c (n_i)! - 1)$ mappings can be pruned so that the computation of $FS(t)$ can be greatly facilitated.

5 The CFFTree Algorithm

In this section, we summarize our CFFTree algorithm, which is short for *Closed Frequent Ftree Mining*. Algorithm 1 illustrates the framework of CFFTree. The algorithm simply calls CF-Mine which recursively mines closed frequent *f*trees of a graph database by a depth-first traversal on the enumeration tree.

Algorithm 2 outlines the pseudo-code of CF-Mine. For each frequent *f*tree t , CFFTree check all candidate frequent *f*tree $t' = t \circ_x v$, to obtain $\mathcal{SO}(t, t', \mathcal{D})$, which is useful to compute $EO(t)$ (Line 1) and $EN(t)$ (Line 2). However, for $t' \in F(t)$, CFFTree only grows t on its extension-frontier, i.e. $t' = t \circ_{ef} v$, which ensures the completeness of frequent *f*trees in \mathcal{D} (Line 7-12). Automorphism-based pruning and canonical mapping-based pruning can be applied to facilitate the computation of the three sets $EO(t)$, $EN(t)$ and $F(t)$. For the frequent *f*tree t , if there exists $t' \in EO_1(t)$, then neither t nor any of t 's descendants in the enumeration tree can be closed, and hence can be efficiently pruned (Line 3-4). If $EO_1(t) = \emptyset$ but there exists $t' \in EO_2(t)$, although we cannot prune t from the enumeration tree, we can apply Theorem 2 to prune some children of t in the enumeration tree (Line 11-12). If $EO(t) = \emptyset$, then no pruning is possible and we have to compute $EN(t)$ to determine the closeness of t , i.e., the naive algorithm mentioned in Section 3 (Line 2). If $EN(t) \neq \emptyset$, t is not closed, otherwise, t

Algorithm 1. *CFFTree* (\mathcal{D}, ϕ)

Input: A graph database \mathcal{D} , the minimum support threshold ϕ

Output: The closed frequent *f*trees set \mathcal{CF}

- 1: $\mathcal{CF} \leftarrow \emptyset$;
 - 2: $\mathcal{F} \leftarrow$ frequent 1-*f*trees;
 - 3: **for all** frequent 1-*f*tree $t \in \mathcal{F}$ **do**
 - 4: $CF\text{-Mine}(t, \mathcal{CF}, \mathcal{D}, \phi)$;
 - 5: **return** \mathcal{CF}
-

Algorithm 2. *CF-Mine* ($t, \mathcal{CF}, \mathcal{D}, \phi$)

Input: A frequent *f*tree t , the set of closed frequent *f*trees, \mathcal{CF} , A graph database \mathcal{D} and the minimum support threshold ϕ

Output: The closed frequent *f*trees set \mathcal{CF}

- 1: Compute $EO(t)$;
 - 2: **if** $EO(t) = \emptyset$ **then** Compute $EN(t)$;
 - 3: **if** $\exists t' \in EO_1(t)$ **then**
 - 4: **return**; // The safe position pruning;
 - 5: **else**
 - 6: $F(t) \leftarrow \emptyset$
 - 7: **for each** equivalence class ec_i on the extension frontier of t **do**
 - 8: **for each** valid vertex v which can be grown on ec_i of t **do**
 - 9: $t' \leftarrow t \circ_{ef} v$, where p , a representative of ec_i , is v 's parent
 - 10: **if** $\text{support}(t') \geq \phi|\mathcal{D}|$ **then**
 - 11: **if** $\nexists t'' \in EO_2(t)$, where $(t'' - t)$ is p and the label of $(t' - t)$ is lexicographically greater than that of $(t'' - t)$ **then**
 - 12: $F(t) \leftarrow F(t) \cup \{t'\}$ // the safe label pruning
 - 13: **for each** frequent t' in $F(t)$ **do**
 - 14: $CF\text{-Mine}(t', \mathcal{CF}, \mathcal{D}, \phi)$
 - 15: **if** $EO(t) = \emptyset$ and $EN(t) = \emptyset$ **then**
 - 16: $\mathcal{CF} \leftarrow \mathcal{CF} \cup \{t\}$
-

is closed (Line 15-16). The set $F(t)$ is computed by extending vertices on the extension frontier of t , which grows the enumeration tree for frequent *f*tree mining (Line 8-12). This procedure proceeds recursively (Line 13-14) until we find all closed frequent *f*trees in the graph database.

6 Experiments

In this section, we report a systematic performance study that validates the effectiveness and efficiency of our closed frequent free tree mining algorithm: **CFFTree**. We use both a real dataset and a synthetic dataset in our experiments. All experiments were done on a 3.4GHz Intel Pentium IV PC with 2GB main memory, running MS Windows XP operating system. All algorithms are implemented in C++ using the MS Visual Studio compiler. We compare **CFFTree** with **F3TM** plus post-processing, thus, the performance curve mainly reflects the effectiveness of pruning techniques mentioned in Section 4.

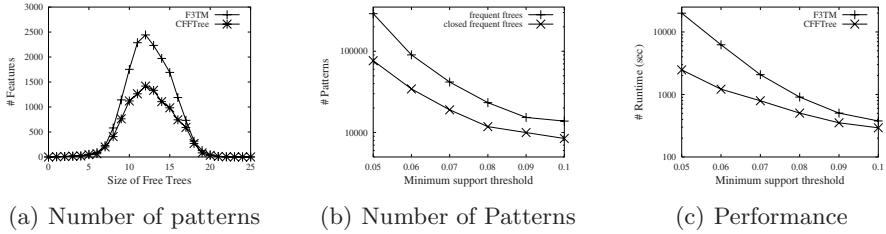


Fig. 7. Mining patterns in real datasets

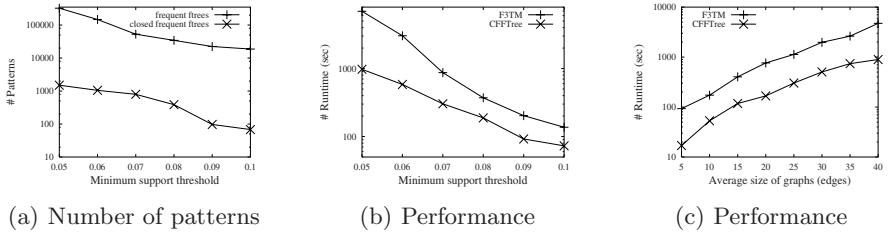


Fig. 8. Mining patterns in synthetic datasets

The real dataset we tested is an AIDS antiviral screen chemical compound database from Developmental Theroapeutics Program in NCI/NIH. The database contains up to 43,905 chemical compounds. There are total 63 kinds of atoms in this database, most of which are *C*, *H*, *O*, *S*, etc. Three kinds of bonds are popular in these compounds: single-bond, double-bond and aromatic-bond. We take atom types as vertex labels and bond types as edge labels. On average, compounds in the database has 43 vertices and 45 edges. The graph of maximum size has 221 vertices and 234 edges.

Figure 7(a) shows the number of frequent patterns w.r.t. the size of patterns (vertex number). We select 10000 chemical compounds from the real database and set the minimum threshold ϕ to be 10%. As shown, most frequent and closed frequent *ftrees* have vertices ranging from 8 to 17. While the number of small *ftrees* with vertex number less than 5 and large *ftrees* with vertex number greater than 20 is quite limited. Figure 7(b) shows the number of patterns of interest with ϕ varying from 5% to 10% and the running time is shown in Figure 7(c) on the same dataset. As we can see, CFFTTree outperforms F3TM by a factor of 10 in average and the ratio between frequent *ftrees* and closed ones is close from 10 to 1.5. It demonstrates that closed pattern mining can deliver more compact mining results.

We then tested CFFTTree on a series of synthetic graph databases, which are generated by the widely-used graph generator [5]. The synthetic dataset is characterized by different parameters, which is described in detail in [5]. Figure 8(a) shows the number of patterns of interest with ϕ varying from 5% to 10% and the running time is shown in Figure 8(b) for the dataset $\mathcal{D}10000I10T30V50$. Compared with the real dataset, CFFTTree has a similar performance gain in this synthetic dataset. We then test the mining performance by changing the

parameter T in the synthetic data, while other parameters keep fixed. The experimental results are shown in Figure 8(c). Again, **CFFTree** performs better than **F3TM**.

7 Conclusion

In this paper, we investigate the problem of *mining closed frequent ftrees from large graph databases*, a critical problem in structural pattern mining because mining all frequent ftrees are inherently inefficient and redundant. Several new pruning algorithms are introduced in this study including the *safe position pruning* and the *safe label pruning* to efficiently prune branches of the search space. The *automorphism-based pruning* and the *canonical mapping-based pruning* are applied in the computation of candidate sets and equivalent occurrence sets, which dramatically facilitate the total mining process. A **CFFTree** algorithm is implemented and our performance study demonstrates its high efficiency over the up-to-date frequent ftree mining algorithms. To our best knowledge, this is the first piece of work on closed frequent ftree mining on large graph databases.

Acknowledgment. This work was supported by a grant of RGC, Hong Kong SAR, China (No. 418206).

References

1. Yun Chi, Yi Xia, Yirong Yang, and Richard R. Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):190–202, 2005.
2. Yun Chi, Yirong Yang, and Richard R. Muntz. Indexing and mining free trees. In *Proceedings of ICDM03*, 2003.
3. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.
4. Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of ICDM03*, 2003.
5. Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of ICDM01*, 2001.
6. Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of KDD04*, 2004.
7. Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proceeding of ICDT99*, 1999.
8. Ulrich Rückert and Stefan Kramer. Frequent free tree discovery in graph data. In *Proceedings of SAC04*, 2004.
9. Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of ICDM02*, 2002.
10. Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns. In *Proceedings of KDD03*, 2003.
11. Xifeng Yan, Jiawei Han, and Ramin Afshar. Clospan: Mining closed sequential patterns in large databases. In *Proceedings of SDM03*, 2003.
12. Peixiang Zhao and Jeffrey Xu Yu. Fast frequent free tree mining in graph databases. In *Proceedings of MCD06 - ICDM 2006 Workshop*, Hong Kong, China, 2006.

Mining Time-Delayed Associations from Discrete Event Datasets*

K.K. Loo and Ben Kao

Department of Computer Science,
The University of Hong Kong, Hong Kong
{kkloo, kao}@cs.hku.hk

Abstract. We study the problem of finding time-delayed associations among types of events from an event dataset. We present a baseline algorithm for the problem. We analyse the algorithm and identify two methods for improving efficiency. First, we propose pruning strategies that can effectively reduce the search space for frequent time-delayed associations. Second, we propose the breadth-first* (BF*) candidate-generation order. We show that BF*, when coupled with the least-recently-used cache replacement strategy, provides a significant saving in I/O cost. Experiment results show that combining the two methods results in a very efficient algorithm for solving the time-delayed association problem.

1 Introduction

Developments in sensor network technology have attracted vast amounts of research interest in recent years [1][2][3][6][7][8][9]. One of the research topics related to sensor networks is to find correlations among the behaviour of different sensors.

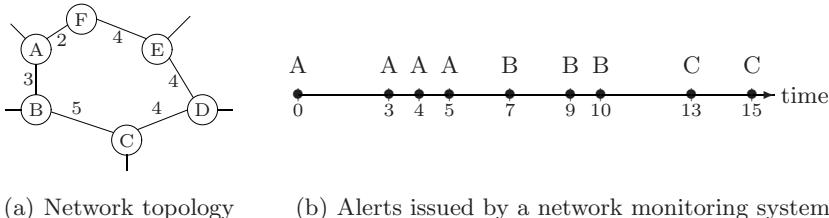


Fig. 1. An example showing a network monitoring system

Consider a network monitoring system designed for collecting traffic data of a network of switches and links as shown in Figure 1(a). In the figure, nodes represent switches, whereas edges are links connecting switches. Under normal conditions, the time needed to pass through a link is represented by the number on the corresponding edge. When the traffic at Switch X exceeds certain capacity, a congestion alert is raised. Figure 1(b) shows an example of alert signals.

* This research is supported by Hong Kong Research Grants Council Grant HKU 7138/04E.

By analysing an alert sequence, one may discover interesting correlations among different types of alerts. For example, one may find that a Switch-A alert is likely followed by a Switch-B alert within a certain time period. One may also find that if such an A-B pattern occurs, a Switch-C alert is likely to occur soon after. Such association information would be useful, for example, in congestion prediction, which could be applied to intelligent traffic redirection strategies.

In this paper, we model correlations of events in the form of *time-delayed associations*. In our model, an event e is a pair (E_e, t_e) where E_e is its type and t_e is the time at which e occurs. We are interested in associations among events whose occurrences are time-constrained. A time-delayed association thus takes the form $I \xrightarrow{[u,v]} J$, where I, J are event types and u, v are two time values such that $0 < u \leq v$. The association captures the observation that when an event i of type I occurs at some time t_i , it is likely that an event j of type J occurs at time t_j such that $t_i + u \leq t_j \leq t_i + v$. If such an event j exists, event i is said to *match* the association and we call j a *consequence* of i w.r.t. the association.

Associations can be “chained” to form longer associations that involve more than two event types. Chained associations can help detecting risk of unfavourable conditions early. Here, we can treat an association $I \xrightarrow{[u,v]} J$ as a *complex event type* \mathbb{I} . An association between a complex event type \mathbb{I} and an event type K has the form $\mathbb{I} \xrightarrow{[u,v]} K$. Intuitively, such an association refers to the observation that if an event of type I occurs and is followed by one or more event of type J within a certain constrained time period, then at least one of the type- J consequences is likely followed by a type- K event within a constrained time period.

In [5], Mannila et al proposed the concept of *episode*, which is an ordered list of events. They proposed the use of minimal occurrences to find episode rules in order to capture temporal relationships between different event types. A *minimal occurrence* of an episode is a time interval $[t_s, t_e)$ such that the episode occurs in the interval but not in any proper sub-interval of $[t_s, t_e)$. Let α and β be two episodes. An *episode rule* has the form $\alpha[w_1] \Rightarrow \beta[w_2]$, which specifies the following relationship: “if α has a minimal occurrence in the interval $[t_s, t_e)$ such that $t_e - t_s \leq w_1$, then there is a minimal occurrence of β in $[t_s, t'_e)$ such that $t'_e - t_s \leq w_2$ ”. The goal is to discover episodes and episode rules that occur frequently in the data sequence.

In a sense, our problem is similar to episode discovery in that we are looking for frequently occurring event sequences. However, we remark that the use of minimal occurrence to define the occurrence of an episode might in some cases fail to reflect the strength of an association. As an example, consider Figure 1(b) again. It is possible that the three type- B events that occur at time $t = 7, 9$ and 10 are “triggered” respectively by the three preceding A 's that occur at $t = 3, 4$ and 5 . Hence, the association $A \rightarrow B$ has occurred three times. However, only one period ($[5, 8)$) is qualified as a minimal occurrence of the episode $A \rightarrow B$. In other words, out of all 4 occurrences of A in the figure, there is only 1 occurrence of the episode $A \rightarrow B$, even though 3 of the A 's have triggered B .

A major difference between our definition of time-delayed association and the episode's minimal occurrence approach is that, under our approach, every event

that matches an association counts towards the association's support. This fairly reflects the strength of correlations among event types. Also, our definition allows the specification of a timing constraint $[u, v]$ between successive event types in an association. This helps removing those associations that are not interesting. For example, if it takes at least 2 time units for a packet to pass through a switch, then any type-B alert that occurs 1 time unit after a type-A alert should not count towards the association $A \rightarrow B$ (See Figure 1). We can thus use the timing constraint to filter false matches. The minimal occurrence approach used in episode does not offer such flexibility.

A straightforward approach to finding all frequent associations is to generate and verify them incrementally. First, we form all possible length-2 associations $X \rightarrow Y$, where X and Y are any event types in the data sequence. We then scan the data to count the associations' supports. Those associations with high supports are considered frequent. Next, for each frequent association $X \rightarrow Y$, we consider every length-3 extension, i.e., we append every event type Z to $X \rightarrow Y$ forming $(X \rightarrow Y) \rightarrow Z$. The support of those length-3 associations are counted and those that are frequent will be used to generate length-4 associations, and so on. The process stops when we can no longer obtain any new frequent sequences. In Section 3 we will show how the above conceptual procedure is implemented in practice. In particular, we show how the computational problem is reduced to a large number of table joins. We call this algorithm the baseline algorithm.

The baseline algorithm is not particularly efficient. We address two methods to improve its efficiency. First, the baseline algorithm extends a frequent association $\mathbb{I} \rightarrow Y$ by considering all possible extensions $(\mathbb{I} \rightarrow Y) \rightarrow Z$. Many of such extensions could be infrequent and the effort spent on counting their supports is wasted. A better strategy is to estimate upper bounds of the associations' supports and discard those that cannot meet the support requirement. Second, as we will explain later, the baseline algorithm generates $(\mathbb{I} \rightarrow Y) \rightarrow Z$ by retrieving and joining the tables associated with two sub-associations, namely, $\mathbb{I} \rightarrow Y$ and $Y \rightarrow Z$. Since the number of such associations and their associated tables is huge, the tables will have to be disk-resident. A caching strategy that can avoid disk accesses as much as possible would thus have a big impact on the algorithm's performance. In this paper we study an interesting coupling effect of a caching strategy and an association-generation order.

The rest of the paper is structured as follows. We give a formal definition of our problem in Section 2. In Section 3, we discuss some properties of time-delayed associations and propose a baseline algorithm for the problem. In Section 4, we discuss the pruning strategies and the caching strategies. We present experiment results in Section 5 and conclude the paper in Section 6.

2 Problem Definition

In this section we define the problem of finding time-delayed associations from event datasets. We define an event e as a 2-tuple (E_e, t_e) where E_e is the event type and t_e is the time e occurs. Let \mathcal{D} denote an event dataset and \mathcal{E} denote the set of all event types that appear in \mathcal{D} . We define a time-delayed association

as a relation between two event types $I, J \in \mathcal{E}$ of the form $I \xrightarrow{[u,v]} J$. We call I the *triggering event type* and J the *consequence event type* of the association. Intuitively, $I \xrightarrow{[u,v]} J$ captures the observation that if an event i such that $E_i = I$ occurs at time t_i , then it is “likely” that there exists an event j so that $E_j = J$ and $t_i + u \leq t_j \leq t_i + v$, where $v \geq u > 0$. The likelihood is given by the *confidence* of the association, whereas the statistical significance of an association is given by its *support*. We will define support and confidence shortly.

For an association $r = I \xrightarrow{[u,v]} J$, an event i is called a *match* of r (or i *matches* r) if $E_i = I$ and there exists another event j such that $E_j = J$ and $t_i + u \leq t_j \leq t_i + v$. The event j here is called a *consequence* of r . We use the notations M_r to denote the set of all matches of r , $q_{r,i}$ to denote the set of all consequences that correspond to a match i of r and $m_{r,j}$ to denote the set of all matches of r that correspond to a consequence j . Also, we define $Q_r = \bigcup q_{r,i} \forall i \in M_r$. That is, Q_r is the set of all events that are consequences of r . The *support* of an association r is defined as the ratio of the number of matching events to the total number of events (i.e., $\frac{|M_r|}{|\mathcal{D}|}$). The *confidence* of r is defined as the fraction $\frac{|M_r|}{|\mathcal{D}_I|}$, where \mathcal{D}_I is the set of all type- I events in \mathcal{D} . We use the notations $supp(r)$ and $conf(r)$ to represent the support and confidence of r , respectively. Finally, the *length* of an association r , denoted by $len(r)$, is the number of event types contained in r .

We can extend the definition to relate more than two event types. Consider an association $r = I \xrightarrow{[u,v]} J$ as a *complex* event type \mathbb{I} , an association between \mathbb{I} and an ordinary event type K is of the form $r' = \mathbb{I} \xrightarrow{[u,v]} K$. Here, \mathbb{I} is the triggering event type and K is the consequence event type. Intuitively, the association says that if an event of type I is followed by one or more event of type J within certain time constraints u and v , then at least one of the J ’s is likely to be followed by a type K event. A match for the association r' is a match i for r such that, for some j where $j \in q_{r,i}$, there exists an event k such that $E_k = K$ and $t_j + u \leq t_k \leq t_j + v$. We say that event k is a consequence of event i w.r.t. the association r' . The support of r' is defined as the fraction of events in D that match r' (i.e., $\frac{|M_{r'}|}{|\mathcal{D}|}$). The confidence of r' is defined as the ratio of the number of events that match r' to the number of events that match r (i.e., $\frac{|M_{r'}|}{|M_r|}$). Given two user-specified thresholds ρ_s and ρ_c and a timing constraint $[u, v]$, the problem of mining time-delayed associations is to find all associations r such that $supp(r) \geq \rho_s$ and $conf(r) \geq \rho_c$.

In our model, we use the same timing constraint $[u, v]$ for all associations. Therefore, we will use a plain arrow “ \rightarrow ” instead of “ $\xrightarrow{[u,v]}$ ” in the rest of the paper when the timing constraint is clear from the context or is unimportant.

3 The Baseline Algorithm

We start this section with two properties based on which the baseline algorithm is designed.

Property 1: If $|\mathcal{D}_I|$, i.e., the number of occurrences of type I events, is smaller than $\rho_s \times |\mathcal{D}|$, then any association of the form $r = I \rightarrow J$ must be infrequent.

```

algorithm BASELINE
1)  $\mathcal{L} := \emptyset; \mathcal{C} := \emptyset n = 2;$ 
2)  $\mathcal{F} := \{\text{all frequent event types}\}$ 
3) foreach frequent  $I \in \mathcal{F}, J \in \mathcal{E}$  do
4)    $\mathcal{C} := \mathcal{C} \cup \{I \rightarrow J\}$ 
5) end-for
6) while  $\mathcal{C} \neq \emptyset$  do
7)    $\mathcal{C}_n := \mathcal{C}; \mathcal{C} := \emptyset$ 
8)   foreach  $r \in \mathcal{C}_n$  do
9)     if  $r = \mathbb{I} \rightarrow J$  is frequent do
10)       $\mathcal{L} := \mathcal{L} \cup \{r\}$ 
11)       $\mathcal{C} := \mathcal{C} \cup \{(\mathbb{I} \rightarrow J) \rightarrow K\} \forall K \in \mathcal{E}$ 
12)    end-if
13)  end-for
14)   $n := n + 1$ 
15) end-while
16) return  $\mathcal{L}$ 

```

Fig. 2. Algorithm BASELINE

$A \xrightarrow{[3,5]} B$		$B \xrightarrow{[3,5]} C$	
m	q	m	q
3	7	9	13
4	7	10	13
4	9	10	15
5	9		
5	10		

$(A \xrightarrow{[3,5]} B) \xrightarrow{[3,5]} C$	
m	q
4	13
5	13
5	15

(a)

(b)

(c)

Fig. 3. $M\text{-}Q$ mappings for various time-delayed associations

Proof: By definition, the set of matches of r must be a subset of \mathcal{D}_I . Hence, $|M_r| \leq |\mathcal{D}_I| < \rho_s \times |\mathcal{D}|$. \square

Property 2: For any associations x and $y = x \rightarrow K$, $\text{supp}(x) \geq \text{supp}(y)$.

Proof: By definition, $M_x \supseteq M_y$. Hence, $\text{supp}(x) \geq \text{supp}(y)$. \square

From Property 2, we know that if an association y is frequent, so is x . In other words, if an association x is not frequent, we do not need to consider any associations that are *right* extensions of x . The baseline algorithm (Figure 2) generates associations based on this observation.

First, the algorithm collects into the set \mathcal{F} all frequent event types (Line 2). The algorithm then maintains two sets: \mathcal{C} is a set of candidate associations which are to be verified, and \mathcal{L} is a set that contains all frequent associations discovered. The set \mathcal{C} is initialized to contain all possible length-2 associations (Lines 3–5). The support of a candidate association r is determined. (We will discuss how to compute the support shortly.) If r is verified to be frequent, we extend r to $r \rightarrow K$ for each event type $K \in \mathcal{E}$ and add them to \mathcal{C} . The algorithm terminates when all candidates are evaluated and no new candidates can be generated.

To compute an association's support, consider an association $r = (\mathbb{I} \rightarrow J) \rightarrow K$. By definition, an event i is a match of r if i is a match of $r_1 = \mathbb{I} \rightarrow J$ and for some consequence j of i , there exists an event k such that $E_k = K$ and $t_j + u \leq t_k \leq t_j + v$. In other words, j is both a consequence of r_1 and a match of $r_2 = J \rightarrow K$. The set of all such events is given by $Q_{r_1} \cap M_{r_2}$. We call this the *connecting set* between r_1 and r_2 . We then have the following properties.

Property 3: For any event $j \in Q_{r_1} \cap M_{r_2}$, every $i \in m_{r_1,j}$ is a match of r and every $k \in q_{r_2,j}$ is a consequence of event i w.r.t. r for every $i \in m_{r_1,j}$.

Proof: By definition, every $i \in m_{r_1,j}$ is a match of r because there exists k such that $t_j + u \leq t_k \leq t_j + v$. Indeed, every $k \in q_{r_2,j}$ fulfills this requirement. Hence, every $k \in q_{r_2,j}$ is a consequence of i w.r.t. r for every $i \in m_{r_1,j}$. \square

Property 4: For any event $j \notin Q_{r_1} \cap M_{r_2}$, $\exists i \in m_{r_1,j}, k \in q_{r_2,j}$ such that i is a match and k is a consequence of i w.r.t. r .

Proof: (i) Any event $j \notin Q_{r_1}$ cannot be a consequence of any $i \in M_{r_1}$ for the association r_1 . So $m_{r_1,j} = \emptyset$. (ii) For any $j \in Q_{r_1}$ but $j \notin M_{r_2}$, $q_{r_2,j} = \emptyset$. \square

Given an association r and a match i of r , we can determine all consequences j of i w.r.t. r . If we put all these match-consequence $i-j$ pairs in a relation, we obtain an $M-Q$ mapping of the association r . Let us consider the network switch example again (Figure 2). If $r = A \xrightarrow{[3,5]} B$, then the matching type-A event at $t = 4$ leads to two consequence type-B events at $t = 7$ and 9. Hence the tuples $\langle 4, 7 \rangle$ and $\langle 4, 9 \rangle$ are in the $M-Q$ mapping of the association. Figures 3(a) and 3(b) show the $M-Q$ mappings of the associations $A \xrightarrow{[3,5]} B$ and $B \xrightarrow{[3,5]} C$, respectively.

By Property 3, given the $M-Q$ mappings for r_1 and r_2 , denoted respectively by T_1 and T_2 , we can derive the $M-Q$ mapping of r by performing an equijoin on T_1 and T_2 so that $T_1.q = T_2.m$, where the join result is projected on $T_1.m$ and $T_2.q$, removing the duplicate tuples in the mapping. Figure 3(c) shows the resultant $M-Q$ mapping of $(A \xrightarrow{[3,5]} B) \xrightarrow{[3,5]} C$. Given the $M-Q$ mapping of an association r , the support $supp(r)$ can be computed by counting the number of distinct elements in the m column. The confidence of r can then be easily determined by the supports of its sub-associations. In this paper, we focus on computing the supports of associations and extracting those that are frequent.

4 Improving the Baseline Algorithm

The baseline algorithm described in the previous section offers a method to find frequent time-delayed associations. In this section, we propose methods to improve the efficiency of the algorithm by investigating two areas, namely the search space for frequent associations and the handling of intermediate results.

4.1 Pruning Strategy

For our problem of mining time-delayed associations, Properties 1 and 2, described in Section 3, are the only base for trimming the search space for frequent time-delayed associations. So, the baseline algorithm takes all possible extensions of a frequent association as candidates. A better strategy would be to estimate an upper-bound for the support of each candidate, without actually joining the $M-Q$ mappings of its sub-associations, and trim those that cannot be frequent.

Multiplicity of consequences. With respect to a time-delayed association, an event can be a consequence of one or more matches. We define, for an association r , the *multiplicity* of a consequence q as the number of matches such that q is a consequence. Getting the multiplicity values is easy. By sorting the $M-Q$ mapping of an association r by the q column, rows for a particular consequence are arranged consecutively. The multiplicity of each consequence q is thus obtained by the number of consecutive rows corresponding to q . Figure 4(a) shows an example. Based on multiplicity, we propose two methods, namely, GlobalK and SectTop, for efficiently identifying candidates that cannot be frequent.

m	q	q	Mult	seg	SectTop Vector	m	q	seg	num_matches
1	3	3	1	1	$\langle 1 \rangle$	7	9	1	0
4	7	7	3	2	$\langle 3, 5 \rangle$	11	13	2	1
5	7			3	$\langle \rangle$			3	1
6	7								
8	10	10	2						
9	10								

(a) $M\text{-}Q$ mapping and multiplicity of consequences for $I \rightarrow J$ (b) SectTop vectors for $I \rightarrow J$ (c) $M\text{-}Q$ mapping and number of matches per segment for $J \rightarrow K$

Fig. 4. Multiplicity of consequences and SectTop

GlobalK. The notion of multiplicity implies that, for an association $r_1 = I \rightarrow J$, the sum of the multiplicities of n distinct consequences gives an upper-bound on the number of matches associated. Given that r_1 is frequent. To verify whether an association $r = (I \rightarrow J) \rightarrow K$ is frequent, we consider r_1 and $r_2 = J \rightarrow K$. It is clear that the connecting set between r_1 and r_2 contains at most $x = \min(|Q_{r_1}|, |M_{r_2}|)$ events, which in turn are associated to at most y matches of r_1 where y is the sum of the top x multiplicity values w.r.t. r_1 . If y is smaller than $\rho_s \times |\mathcal{D}|$, r cannot be frequent. In general, for a time-delayed association $r_1 = I \rightarrow J$, we call the minimum number of consequences such that the sum of their multiplicity values is not smaller than the support threshold the *GlobalK threshold* for the association. Any extension of r_1 of the form $(I \rightarrow J) \rightarrow K$ cannot be frequent if $|M_{J \rightarrow K}|$ is smaller than the GlobalK threshold of r_1 .

SectTop. GlobalK is a simple method for pruning candidates that cannot be frequent. However, the GlobalK threshold, which is derived from the highest multiplicities for an association, can be too generous as a pruning condition as those consequences with top multiplicities may not all enter the connecting set.

We address this issue in *SectTop*. In simple words, we conceptually divide the whole length of time represented by \mathcal{D} into a number of segments. For a frequent association $r_1 = I \rightarrow J$, for each segment, we capture information on the multiplicities of the consequences occurring within the segment in a *SectTop vector*. Then, when checking whether an association $r = (I \rightarrow J) \rightarrow K$ can be frequent, for each segment, we get an upper-bound on the number of matches that associate to the consequences in the segment. The sum of the upper-bounds for each segment thus gives an overall upper-bound on the number of matches of r . If the overall upper-bound is smaller than $\rho_s \times |\mathcal{D}|$, then r cannot be frequent.

For an association r_1 , the SectTop vector for a segment is obtained as follows. First, the multiplicities for the consequences of r_1 occurring within the segment are sorted inversely. Then, we keep the x highest multiplicity values such that x is minimum and the sum of the x multiplicities exceeds $\rho_s \times |\mathcal{D}|$. If the sum of all x values does not exceed $\rho_s \times |\mathcal{D}|$, we keep all multiplicities. The multiplicity values are then transformed to a vector such that the y -th element is the sum of the top- y multiplicities in the segment. Figure 4(b) shows the SectTop vectors derived from the $M\text{-}Q$ mapping in Figure 4(a) if the length of time represented by the dataset is divided in segments of 5 time units.

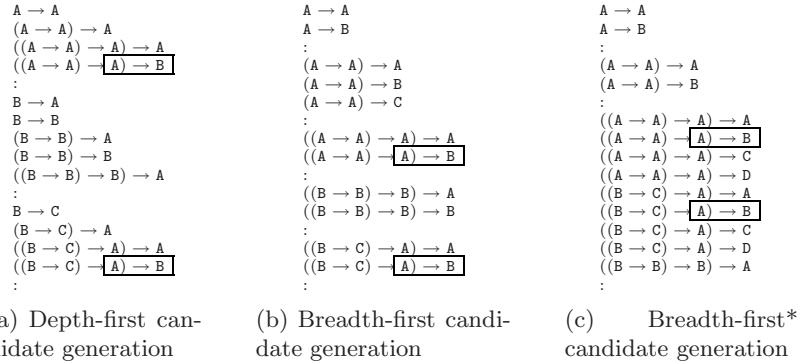


Fig. 5. Candidate generation schemes

To check whether an association $r = (I \rightarrow J) \rightarrow K$ can be frequent, we count the number of distinct matches for the sub-association $r_2 = J \rightarrow K$ appearing in each segment. If there are y matches for $J \rightarrow K$ in the i -th segment, then, in the segment, at most y consequences of r_1 may appear in the connecting set. Hence, an upper-bound on the number of matches associated to the consequences is given by the y -th element of the SectTop vector for the segment. We get the upper-bounds for each segment and their sum gives an overall upper-bound on the number of matches of r . As an example, the table on the left in Figure 4(c) is the M - Q mapping of $J \rightarrow K$, while that on the right lists the number of matches of $J \rightarrow K$ appearing in each segment. When evaluating whether $r = (I \rightarrow J) \rightarrow K$ can be frequent, we check the number of matches of $J \rightarrow K$ in each segment against the SectTop vectors of $I \rightarrow J$. It turns out that the overall upper-bound on the number of matches for r is $(0 + 3 + 0) = 3$. If the support threshold is 4 matches, then we know immediately that r cannot be frequent.

4.2 Cache Management

The baseline algorithm generates a lot of associations during execution. Some of them are repeatedly used for evaluating other candidates later on. Because the volume of data being processed is often very large, keeping all such associations in main memory is not feasible. Maintaining a cache is thus a compromise so that, while keeping some of the intermediate results in memory and reduce I/O accesses, memory can be made available for other operations.

When the cache overflows, part of the cached data is replaced by data fetched from disk. Two commonly used strategies for choosing data for replacement are “Least recently used” (LRU), i.e., data that have not been accessed for the longest time are replaced, and “Least frequently used” (LFU), i.e., least frequently accessed data in the cache are replaced.

The effectiveness of cache, i.e., the likeliness that the data accessed is in the cache, is often mentioned as the *hit-rate*. We argue that the hit-rate is related to the order that candidates are evaluated and the cache replacement strategy

chosen. Figure 5(a) and (b) shows two candidate generation schemes commonly used in level-wise algorithms. Figure 5(a) illustrates a depth-first (DF) candidate generation, i.e., after an association $r = I \rightarrow J$ is evaluated as frequent, the algorithm immediately generates candidates by extending r and evaluates them. Figure 5(b) illustrates a breadth-first (BF) candidate generation that all candidates of the same length are evaluated before longer candidates. These candidate generation schemes would not work well with the LRU strategy. For example, in Figure 5(a), $A \rightarrow B$ is referenced when evaluating the candidates $((A \rightarrow A) \rightarrow A) \rightarrow B$ and $((B \rightarrow C) \rightarrow A) \rightarrow B$. Between the accesses, a number of other candidates are evaluated, which means that many different associations are brought into the memory and cache overflows are more likely. When $A \rightarrow B$ is accessed the second time, its $M-Q$ mapping may no longer reside in the cache. Similar problem exists in the BF scheme (see Figure 5(b)).

It is noteworthy that, in the baseline algorithm, length-2 associations are repeatedly referenced for candidate evaluation. In particular, when evaluating extensions of an association $I \rightarrow J$, each of length-2 associations of the form $J \rightarrow K$ is referenced. By processing as a batch all associations in L_i with the same consequence event type (see Figure 5(c)), we ensure that length-2 associations of the form $J \rightarrow K$ are accessed closely temporally, which favours the LRU strategy. This observation can be easily fitted into the BF candidate generation scheme. At the end of each iteration, we sort the associations in L_i by their consequence event type. Then the sorted associations are fetched sequentially for candidate generation. We call this the *breadth-first** (BF*) candidate generation scheme.

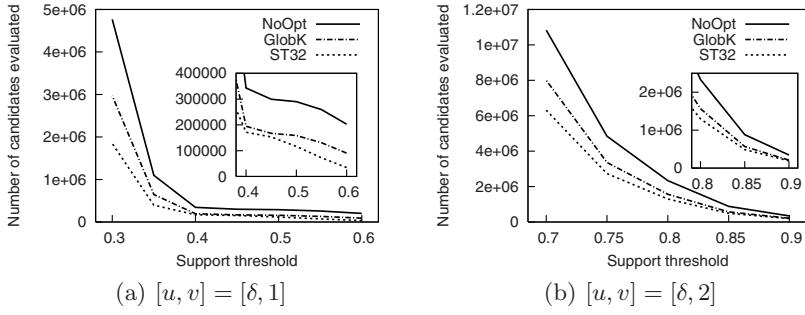
5 Experiment Results

We conducted experiments using stock price data. Due to space limitation, we leave the discussion on how the raw data is transformed into an event dataset in [4]. The transformed dataset consists 99 event types and around 45000 events.

5.1 Pruning Strategy

In the first set of experiments, we want to study the effectiveness of the pruning strategy “GlobalK” and “SectTop”. The effectiveness is best reflected by the number of candidate associations being evaluated. Figure 6 shows the number of candidate associations evaluated when ρ_s is set at different values. We comment that a candidate is regarded as “evaluated” only if the $M-Q$ mapping of the candidate is enumerated. The lines labelled “NoOpt”, “GlobalK” and “ST32” represent respectively the case that no pruning strategy (i.e., the original baseline algorithm) is used, that “GlobalK” is chosen and that “SectTop” is chosen with the time covered by \mathcal{D} divided into 32 segments.

Figure 6(a) shows the results when u and v are set to δ (i.e., a value just bigger than 0) and 1 respectively. As shown in the figure, both GlobalK and SectTop save a major fraction of candidate evaluations performed. At high support (0.6%), savings of 55% and 82% are observed respectively with GlobalK and SectTop over the baseline algorithm while, at low support (0.3%), the savings are

**Fig. 6.** No. of candidates evaluated at different ρ_s

32% and 63%. Similar trend is observed when we changed v to 2 (Figure 6(b)). Although the savings are not as dramatic as in the case when $v = 1$, at low support (0.7%), GlobalK and SectTop achieve savings of 26% and 41%, while at high support (0.9%), the savings are around 39% and 44% respectively.

As shown by the figures, SectTop always outperforms GlobalK in terms of number of candidates being evaluated. A reason is that, for each candidate c , SectTop obtains an upper-bound on $supp(c)$ by estimating the number of matches that are associated to the consequences in each segment. A reasonably fine segmentation of the time covered by \mathcal{D} thus ensures that the upper-bound obtained is relatively tight. For GlobalK, however, the GlobalK threshold for a frequent association is calculated from the highest multiplicity values without considering where these values actually exist in the whole period of time covered by \mathcal{D} . So, the pruning ability of GlobalK is not as good as that of SectTop.

5.2 Candidate Generation, Cache Replacement Strategy and I/O Costs

In the second set of experiments, we want to study the effect of candidate generation orders on different cache replacement strategies. We plot the number of $M\text{-}Q$ mapping tuples read from disk, reflecting total I/O requirement, against the size of the cache in Figures 7 and 8.

We start the analysis with the LRU strategy and ρ_s set to a relative low value at 0.3%. Figure 7(a) shows the I/O performance when no pruning strategy is applied. From the figure, we find that the I/O performance of breadth-first and that of depth-first strategies are very close to each other. For BF* strategy, the I/O cost begins to drop at 16000 tuples and then drops dramatically. The improvement levels down when the cache size is increased to 24000 tuples.

The sharp improvement here is no coincidence. Recall that in BF* candidate generation, at the end of each iteration, we sort the newly found frequent associations by their consequence event type. Candidates are formed and evaluated by extending each of the sorted associations sequentially. In other words, after candidates of the form $(\mathbb{I}_1 \rightarrow J) \rightarrow K$ are evaluated (for some simple or complex event type \mathbb{I}_1), next evaluated are those of the form $(\mathbb{I}_2 \rightarrow J) \rightarrow K$, if such \mathbb{I}_2 exists. The whole set of length-2 associations with triggering event type J are

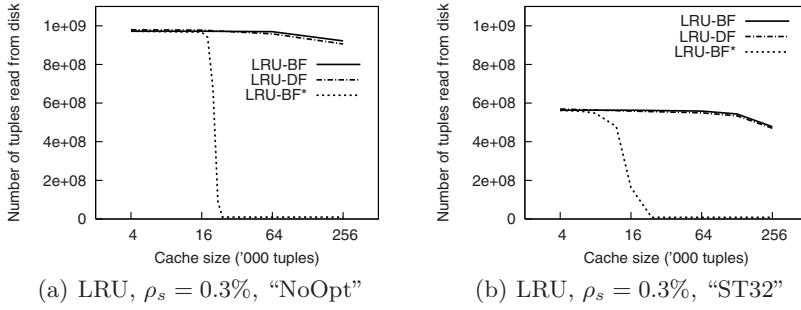


Fig. 7. I/O requirement for LRU ($[u, v] = [\delta, 1]$, $\rho_s = 0.3\%$)

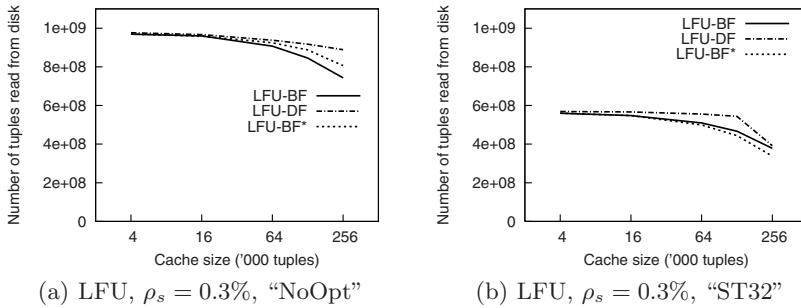


Fig. 8. I/O requirement for LFU ($[u, v] = [\delta, 1]$, $\rho_s = 0.3\%$)

accessed multiple times for these candidates. If the cache is big enough to hold the $M\text{-}Q$ mappings of all such length-2 associations, it is likely that the $M\text{-}Q$ mappings are in the cache after they are referenced for the first time. For the dataset used in the experiment, we find that the maximum sum of the sizes of all $M\text{-}Q$ mappings of a particular triggering event type is about 22000 tuples. A cache with 24000-tuple capacity is thus big enough to save most I/O accesses.

Figure 7(b) shows the case when "ST32" is applied. The curves are similar in shape compared to those in the "NoOpt" case. A big drop in I/O access is also observed with the curve of BF* and the big drop begins at the cache size of 10000 tuples. This is because SectTop avoids evaluating candidates that cannot be frequent. So, for a frequent association $\mathbb{I} \rightarrow J$, it is not necessary to evaluate every candidate of the form $(\mathbb{I} \rightarrow J) \rightarrow K$. A smaller cache is thus enough to hold the $M\text{-}Q$ mappings of length-2 associations used for candidate evaluation.

Figure 8 shows the case of LFU. From the figure, all three candidate generation methods are very similar in terms of I/O requirement. Both depth-first and breadth-first generation performed slightly better when LFU was adopted instead of LRU. However, the "big drop" with BF* is not observed and so the performance of BF* is much worse than the case with LRU. It is because the LFU strategy gives preference to data that are frequently accessed when deciding on what to keep in the cache. This does not match the idea of BF* candidate generation, which works best when recently accessed data are kept in the cache.

In addition, associations entered the cache early may reside in the cache for a long time because, when they are first used for evaluating candidates, a certain number of accesses have been accumulated. Associations newly added to the cache must be accessed even more frequently to stay in the cache.

6 Conclusion

We propose time-delayed association as a way to capture time-delayed dependencies between types of events. We illustrate how time-delayed associations can be found from event datasets in a simple baseline algorithm.

We identify in the simple algorithm two areas for improvement. First, we can get upper-bounds on the supports of candidate associations. Those that cannot be frequent are discarded without finding their actual supports. We proposed two methods, namely, GlobalK and SectTop, for getting an upper-bound on a candidate's support. Experiment results show that these methods reduce significantly the number of candidates being evaluated.

Second, some of the intermediate results generated are repeatedly used for candidate evaluation. Since the volume of data being processed is likely to be high, such intermediate results must be disk-resident and are brought into main memory only when needed. Caching of the intermediate results is thus important for reducing expensive I/O accesses. We find that the order that candidate associations are formed and evaluated would affect the performance of the cache. Experiment results show that the BF* candidate generation scheme, coupled with a reasonably-sized cache and the LRU cache replacement strategy, can comprehensively reduce the I/O requirement of the algorithm.

References

1. Xiaonan Ji, James Bailey, and Guozhu Dong. Mining minimal distinguishing subsequence patterns with gap constraints. In *ICDM*, pages 194–201, 2005.
2. Daesu Lee and Wonsuk Lee. Finding maximal frequent itemsets over online data streams adaptively. In *ICDM*, pages 266–273, 2005.
3. Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. Semantics and evaluation techniques for window aggregates in data streams. In *SIGMOD Conference*, pages 311–322, 2005.
4. K. K. Loo and Ben Kao. Mining time-delayed associations from discrete event datasets. Technical Report TR-2007-01, Department of Computer Science, The University of Hong Kong, Hong Kong, 2007.
5. Heikki Mannila and Hannu Toivonen. Discovering generalized episodes using minimal occurrences. In *KDD*, pages 146–151, 1996.
6. Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005.
7. Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. Braid: Stream mining through group lag correlations. In *SIGMOD Conference*, pages 599–610, 2005.
8. Mohammed Javeed Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.
9. Rui Zhang, Nick Koudas, Beng Chin Ooi, and Divesh Srivastava. Multiple aggregations over data streams. In *SIGMOD Conference*, pages 299–310, 2005.

A Comparative Study of Ontology Based Term Similarity Measures on PubMed Document Clustering

Xiaodan Zhang¹, Liping Jing², Xiaohua Hu¹, Michael Ng³, and Xiaohua Zhou¹

¹ College of Information Science & Technology, Drexel University, 3141 Chestnut,
Philadelphia, PA 19104, USA

² ETI & Department of Math, The University of Hong Kong, Pokfulam Road, Hong Kong

³ Department of Mathematics, Hong Kong Baptist University, Kowloon Tong, Hong Kong
{xzhang, thu}@ischool.drexel.edu, lipingjing@eti.hku.hk,
mng@math.hkbu.edu.hk, xiaohua.zhou@drexel.edu

Abstract. Recent research shows that ontology as background knowledge can improve document clustering quality with its concept hierarchy knowledge. Previous studies take term semantic similarity as an important measure to incorporate domain knowledge into clustering process such as clustering initialization and term re-weighting. However, not many studies have been focused on how different types of term similarity measures affect the clustering performance for a certain domain. In this paper, we conduct a comparative study on how different semantic similarity measures of term including path based similarity measure, information content based similarity measure and feature based similarity measure affect document clustering. We evaluate term re-weighting as an important method to integrate domain ontology to clustering process. Meanwhile, we apply k-means clustering on one real-world text dataset, our own corpus generated from PubMed. Experiment results on 8 different semantic measures have shown that: (1) there is no a certain type of similarity measures that significantly outperforms the others; (2) Several similarity measures have rather more stable performance than the others; (3) term re-weighting has positive effects on medical document clustering, but might not be significant when documents are short of terms.

Keywords: Semantic Similarity Measure, Document Clustering, Domain Ontology.

1 Introduction

Recent research has been focused on how to integrate domain ontology as background knowledge to document clustering process and shows that ontology can improve document clustering performance with its concept hierarchy knowledge [2, 3, and 16]. Hotho et al. [2] uses WordNet synsets to augment document vector and achieves better results than that of “bag of words” model on public domain. Yoo et al. [16] achieves promising cluttering result using MeSH domain ontology for clustering initialization. They first cluster terms by calculating term semantic similarity using MeSH ontology (<http://www.nlm.nih.gov/mesh/>) on PubMed document sets [16].

Then the documents are mapped to the corresponding term cluster. Last, mutual reinforcement strategy is applied. Varelas et al. [14] uses term re-weighting for information retrieval application. Jing et al. [3] adopt similar technique on document clustering. They re-weight terms and assign more weight to terms that are more semantically similar with each other.

Although existing approaches rely on term semantic similarity measure, not many studies have been done on evaluating the effects of different similarity measures on document clustering for a specific domain. Yoo et al. [16] uses only one similarity measure that calculates the number of shared ancestor concepts and the number of co-occurred documents. Jing et al. [3] compares two ontology based term similarity measure. Even though these approaches are heavily relied on term similarity information and all these similarity measures are domain independent, however, to date, relatively little work has been done on developing and evaluating measures of term similarity for biomedical domain (where there are a growing number of ontologies that organize medical concepts into hierarchies such as MeSH ontology) on document clustering.

Clustering initialization and term re-weighting are two techniques adopted for integrating domain knowledge. In this paper, term re-weighting is chosen because: (1) a document is often full of class-independent “general” terms, how to discount the effect of general terms is a central task. Term re-weighting may help discount the effects of class-independent general terms and aggravate the effects of class-specific “core” terms; (2) hierarchically clustering terms [16] for clustering initialization is more computational expensive and more lack of scalability than that of term re-weighting approach.

As a result, in this paper, we evaluate the effects of different term semantic similarity measures on document clustering using term re-weighting, an important measure for integration domain knowledge. We examine 4 path based similarity measures, 3 information content based similarity measures, and 2 feature based similarity measures for document clustering on PubMed document sets. The rest of the paper is organized as follows: Section 2 describes term semantic similarity measures; section 3 shows document representation and defines the term re-weighting scheme. In section 4, we present and discuss experiment results. Section 5 concludes the paper shortly.

2 Term Semantic Similarity Measure

Ontology based similarity measure has some advantages over other measures. First, ontology is created by human being manually for a domain and thus more precise; second, compared to other methods such as latent semantic indexing, it's much more computational efficient; Third, it helps integrate domain knowledge into the data mining process. Comparing two terms in a document using ontology information usually exploit the fact that their corresponding concepts within ontology usually have properties in the form of attributes, level of generality or specificity, and their relationships with other concepts [11]. It should be noted that there are many other term semantic similarity measures such as latent semantic indexing, but it's out of

scope of our research, our focus here is on term semantic similarity measure using ontology information. In the subsequent subsections, we classify the ontology based semantic measures into the following three categories and try to pick popular measures for each category.

2.1 Path Based Similarity Measure

Path based similarity measure usually utilizes the information of the shortest path between two concepts, of the generality or specificity of both concepts in ontology hierarchy, and of their relationships with other concepts.

Wu and Palmer [15] present a similarity measure finding the most specific common concept that subsumes both of the concepts being measured. The path length from most specific shared concept is scaled by the sum of IS-A links from it to the compared two concepts.

$$S_{W\&P}(C_1, C_2) = \frac{2H}{N_1 + N_2 + 2H} \quad (1)$$

In the equation (1), N_1 and N_2 is the number of IS-A links from C_1, C_2 respectively to the most specific common concept C , and H is the number of IS-A links from C to the root of ontology. It scores between 1(for similar concepts) to 0. In practice, we set H to 1 when the parent of the most specific common concept C is the root node.

Li et al. [8] combines the shortest path and the depth of ontology information in a non-linear function:

$$S_{Li}(C_1, C_2) = e^{-\alpha L} \frac{e^{\beta H} - e^{-\beta H}}{e^{\beta H} + e^{-\beta H}} \quad (2)$$

where L stands for the shortest path between two concepts, α and β are parameters scaling the contribution of shortest path length and depth respectively. The value is between 1(for similar concepts) and 0. In our experiment, the same as [8]'s, we set α and β to 0.2 and 0.6 respectively.

Leacock and Chodorow [7] define a similarity measure based on the shortest path $d(C_1, C_2)$ between two concepts and scaling that value by twice the maximum depth of the hierarchy, and then taking the logarithm to smooth the resulting score:

$$S_{L\&C}(C_1, C_2) = -\log(d(C_1, C_2)/2D) \quad (3)$$

where D is the maximum depth of the ontology and similarity value. In practice, we add 1 to both $d(C_1, C_2)$ and $2D$ to avoid $\log(0)$ when the shortest path length is 0.

Mao et al. [10] define a similarity measure using both shortest path information and number of descendants of compared concepts.

$$S_{Mao}(C_1, C_2) = \frac{\delta}{d(C_1, C_2) \log_2(1 + d(C_1) + d(C_2))} \quad (4)$$

where $d(C_1, C_2)$ is the number of edges between C_1 and C_2 , $d(C_1)$ is the number of C_1 's descendants, which represents the generality of the concept. Here, the constant

δ refers to a boundary case where C_1 is the only direct hypernym of C_2 , C_2 is the only direct hyponym of C_1 and C_2 has no hyponym. In this case, because the concepts C_1 and C_2 are very close, δ should be chosen close to 1. In practice, we set it to 0.9.

2.2 Information Content Based Measure

Information content based measure associates probabilities with concepts in the ontology. The probability [11] is defined in equation (5), where $freq(C)$ is the frequency of concept C , and $freq(Root)$ is the frequency of root concept of the ontology. In this study, the frequency count assigned to a concept is the sum of the frequency counts of all the terms that map to the concept. Additionally, the frequency counts of every concept includes the frequency counts of subsumed concepts in an IS-A hierarchy.

$$IC(C) = -\log \left(\frac{freq(C)}{freq(Root)} \right) \quad (5)$$

As there may be multiple parents for each concept, two concepts can share parents by multiple paths. We may take the minimum $IC(C)$ when there is more than one shared parents, and then we call concept C the most informative subsumer— $IC_{mis}(C_1, C_2)$. In another word, $IC_{mis}(C_1, C_2)$ has the least probability among all shared subsumer between two concepts.

$$S_{Resnik}(C_1, C_2) = -\log IC_{mis}(C_1, C_2) \quad (6)$$

$$S_{Jiang}(C_1, C_2) = -\log IC(C_1) - \log IC(C_2) + 2\log IC_{mis}(C_1, C_2) \quad (7)$$

Resnik [12] presents a similarity measure. It signifies that the more information two terms share in common, the more similar they are, and the information shared by two terms is indicated by the information content of the term that subsume them in the ontology. The measure reveals information about the usage within corpus of the part of the ontology queried. Jiang [4] includes not only the shared information content between two terms, but also the information content each term contains.

Lin [9] utilizes both the information needed to state the commonality of two terms and the information needed to fully describe these two terms. Since $IC_{mis}(C_1, C_2) \geq \log IC(C_1) + \log IC(C_2)$ the similarity value varies between 1(for similar concepts) and 0.

$$S_{Lin}(C_1, C_2) = \frac{2\log IC_{mis}(C_1, C_2)}{\log IC(C_1) + \log IC(C_2)} \quad (8)$$

2.3 Feature Based Measure

Feature based measure assumes that each term is described by a set of terms indicating its properties or features. Then, the more common characteristics two terms have and the less non-common characteristics they have, the more similar the

terms are [14]. As there is no describing feature set for MeSH descriptor concepts, in our experimental study, we take all the ancestor nodes of each compared concept as their feature sets. The following measure is defined according to [5, 9]:

$$S_{BasicFeature}(C_1, C_2) = \frac{|Ans(C_1) \cap Ans(C_2)|}{|Ans(C_1) \cup Ans(C_2)|} \quad (9)$$

where $Ans(C_1)$ and $Ans(C_2)$ correspond to description sets (the ancestor nodes) of terms C_1 and c_2 respectively, $C_1 \cap C_2$ is the join of two parent node sets and $C_1 \cup C_2$ is the union of two parent node sets.

Knappe [5] defines a similarity measure as below using the information of generalization and specification of two compared concepts:

$$S_{Knappe}(C_1, C_2) = p \times \frac{|Ans(C_1) \cap Ans(C_2)|}{|Ans(C_1)|} + (1-p) \times \frac{|Ans(C_1) \cap Ans(C_2)|}{|Ans(C_2)|} \quad (10)$$

where p 's range is [0, 1] that defines the relative importance of generalization vs. specialization. This measure scores between 1 (for similar concepts) and 0. In our experiment, p is set to 0.5.

3 Document Representation and Re-weighting Scheme

MeSH. Medical Subject Headings (MeSH) mainly consists of the controlled vocabulary and a MeSH Tree. The controlled vocabulary contains several different types of terms, such as Descriptor, Qualifiers, Publication Types, Geographics, and Entry terms. Among them, Descriptors and Entry terms are used in this study since they are terms that can be extracted from documents. Descriptor terms are main concepts or main headings. Entry terms are the synonyms or the related terms to descriptors. For example, “Neoplasms” as a descriptor has the following entry terms {“Cancer”, “Cancers”, “Neoplasm”, “Tumors”, “Tumor”, “Benign Neoplasm”, “Neoplasm, Benign”}. MeSH descriptors are organized in a MeSH Tree, which can be seen as the MeSH Concept Hierarchy. In the MeSH Tree there are 15 categories (e.g. category A for anatomic terms), and each category is further divided into subcategories. For each subcategory, corresponding descriptors are hierarchically arranged from most general to most specific. In addition to its ontology role, MeSH descriptors have been used to index MEDLINE articles. For this purpose, about 10 to 20 MeSH terms are manually assigned to each article (after reading full papers). On the assignment of MeSH terms to articles, about 3 to 5 MeSH terms are set as “MajorTopics” that primarily represent an article.

With mesh descriptor and MeSH tree, the similarity score between two medical terms can be easily calculated. Therefore, we first match the terms in each document abstract to the Entry terms in MeSH and then maps the selected Entry terms into MeSH Descriptors. We select those candidate terms (1-6gram) that only match with MeSH Entry terms. We then replace those semantically similar Entry terms with the Descriptor term to remove synonyms. We next filter out some MeSH Descriptors that are too general (e.g. HUMAN, WOMEN or MEN) or too common in MEDLINE articles (e.g. ENGLISH ABSTRACT or DOUBLE-BLIND METHOD). We assume

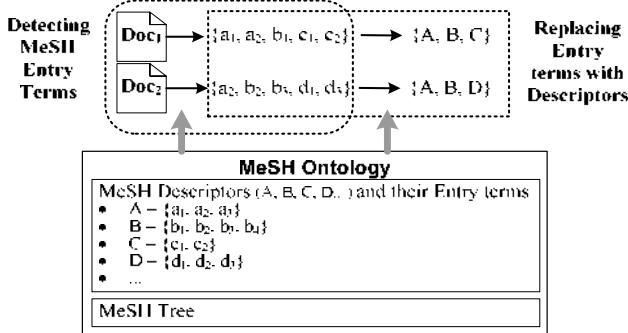


Fig. 1. The concept mapping from MeSH entry terms to MeSH descriptors

that those terms do not have distinguishable power in clustering documents. Hence, we have selected a set of only meaningful corpus-level concepts, in terms of MeSH Descriptors, representing the documents. We call this set *Document Concept Set (DCS)*, where $DCS = \{C_1, C_2, \dots, C_n\}$ and C_i is a corpus-level concept. Fig.1 shows that MeSH Entry term sets are detected from “*Doc₁*” and “*Doc₂*” documents using the MeSH ontology, and then the Entry terms are replaced with Descriptors based on the MeSH ontology. For a more comprehensive comparative study, we represent document in two ways: MeSH entry terms, MeSH descriptor terms. At the time of this writing, there are about 23833 unique MeSH descriptor terms, 44978 MeSH ontology nodes (one descriptor term might belong to more than one ontology nodes) and 593626 MeSH entry terms.

Re-weighting Scheme. A document is often full of class-independent “general” words and short of class-specific “core” words, which leads to the difficulty of document clustering. Steinbach et al. [13] examines on the data that each class has a “core” vocabulary of words and remaining “general” words may have similar distributions on different classes. To solve this problem, we should “discount” general words and “emphasize” more importance on core words in a vector [17]. [3, 14] define the term re-weighting scheme as below

$$\tilde{x}_{ji1} = x_{ji1} + \sum_{\substack{i_2=1 \\ i_2 \neq i_1 \\ S(x_{ji1}, x_{ji2}) \geq \text{Threshold}}}^m S(x_{ji1}, x_{ji2}) \cdot x_{ji2} \quad (11)$$

where x stands for term weight, m stands for the number of co-occurred terms, and $S(x_{ji1}, x_{ji2})$ stands for the semantic similarity between two concepts. Through this re-weighting scheme, the weights of semantically similar terms will be co-augmented. Here the threshold stands for minimum similarity score between two compared terms. Since we are only interested in re-weighting those terms that are more semantically similar with each other, it's necessary to set up a threshold value—the minimum similarity score between compared terms. Besides, it should be noted that the term weight can be referred as term frequency (TF), normalized term frequency (NTF) and TF*IDF (Inverse Document Frequency).

4 Experiment Setting and Result Analysis

4.1 Datasets and Indexing Schemes

We conduct experiments on public MEDLINE documents (abstracts). First we collect document sets related to various diseases from MEDLINE. We use “MajorTopic” tag along with the disease-related MeSH terms as queries to MEDLINE. Table 1 shows the 10 document sets (24566 documents) retrieved from MEDLINE. Then, the collected dataset is indexed using two schemes: *MeSH entry term* and *MeSH descriptor term*. The average document length for MeSH entry term and MeSH descriptor are 14 and 13 respectively (as shown in table 2). Compared to the average document length—81 when using bag of words representation, the dimension of clustering space is dramatically reduced. A general stop word list is applied to bag of words scheme. Moreover, we collect PubMed documents from 1995-2005 to make MeSH descriptor stop term list for MeSH term and MeSH descriptor term indexing. Since a MeSH entry term can be mapped to more than one MeSH descriptor term in MeSH ontology, we then map it to the MeSH descriptor term which is semantically similar with most of the other terms in the document. For a better comparative study, we also make the following environmental settings: 1) the number of clusters is set to 10, the same as the number of the document sets; 2) documents with length less than 5 are removed from the clustering process; 3) when conducting k-means clustering, we run ten times with random initialization and take the average as the result. During the comparative experiment, each run has the same initialization.

4.2 Evaluation Methodology

Cluster quality is evaluated by four extrinsic measures, *entropy* [13], *F-measure* [6], *purity* [19], and *normalized mutual information (NMI)* [1]. Because of space restrictions, we only describe in detail a recently popular measure—NMI, which is defined as the mutual information between the cluster assignments and a pre-existing labeling of the dataset normalized by the arithmetic mean of the maximum possible entropies of the empirical marginal, i.e.,

$$NMI(X, Y) = \frac{I(X; Y)}{(\log k + \log c)/2} \quad (12)$$

where X is a random variable for cluster assignments, Y is a random variable for the pre-existing labels on the same data, k is the number of clusters, and c is the number of pre-existing classes. NMI ranges from 0 to 1. The bigger the NMI is the higher quality the clustering is. NMI is better than other common extrinsic measures such as purity and entropy in the sense that it does not necessarily increase when the number of clusters increases. For *Purity* and *F-measure* ranging from 0 to 1, the bigger the value is the higher quality the clustering has. For entropy, the smaller the value is the higher clustering quality is.

Table 1. The Document Sets and Their Sizes

	Document Sets	No. of Docs
1	Gout	642
2	Chickenpox	1,083
3	Raynaud Disease	1,153
4	Jaundice	1,486
5	Hepatitis B	1,815
6	Hay Fever	2,632
7	Kidney Calculi	3,071
8	Age-related Macular Degeneration	3,277
9	Migraine	4,174
10	Otitis	5,233

Table 2. Document indexing schemes

Indexing Scheme	No. of term indexed	Avg. doc length
MeSH entry term	14885	14
MeSH descriptor term	8829	13
Word	41208	81

4.3 Result Analysis

To compare the effects of different similarity measures on improving clustering quality, we run k-means clustering on the collected dataset. We represent each document as TF*IDF vector, because this scheme achieves much better performance than NTF and TF. Cosine similarity measure is applied when calculating the distance between one document vector and the cluster center vector. Moreover, when representing a document using MeSH entry terms, it's somewhat similar with augmenting a document vector with synonym terms. As one MeSH descriptor term can relate with many different MeSH entry terms, it is possible that two or more MeSH entry terms with same descriptor term appear in one document. Furthermore, if a document is represented as a document using MeSH descriptors, it can help map all the synonyms occurred in one document to their according descriptor terms. In this paper, we evaluate the clustering qualities of both representation schemes as well as word representation scheme. The process of clustering is as follows: (1) index the document sets using MeSH entry terms or MeSH descriptor terms; (2) calculate term similarity using selected similarity measure and then build similarity matrix for indexed terms; (3) re-weight terms in each document vector using similarity matrix and equation (10); (4) Run k-means clustering. We use dragon toolkit [18] to implement the whole process.

Experimental results show that of the three types of term similarity measures, there is no a certain type of measures that significantly outperforms others. This can be partially resulted from the fact that most of these measures consider not only the term closeness within the ontology but also the depth of the two compared concepts within the ontology. Apparently, the similarity score of $S_{L\&C}$, S_{Resnik} and S_{Jiang} is not within

Table 3. Clustering results of MeSH entry terms scheme; each measure is followed by the threshold of similarity value (in parenthesis) that helps achieve the best results

Type of Measure	Similarity Measure	Entropy	F-Score	Purity	NMI
Path based	Wu & Palmer (0.8)	0.392	0.803	0.876	0.757
	Li et al. (0.7)	0.353	0.830	0.871	0.771
	Leacock (0.2)	0.930	0.596	0.686	0.524
	Mao et al. (0.8)	0.338	0.836	0.885	0.781
Information Content	Resnik (0.0)	0.353	0.821	0.877	0.774
	Jiang (0.1)	0.572	0.695	0.799	0.701
	Lin (0.9)	0.360	0.825	0.880	0.771
Feature based	Basic Feature (0.8)	0.389	0.795	0.874	0.759
	Knappe (0.8)	0.484	0.778	0.831	0.717
MeSH entry term	None	0.363	0.800	0.870	0.774
Word	None	0.245	0.755	0.908	0.820

[0, 1]. So term similarity scores using these three measures are normalized before being applied to do term reweighting for a fair comparison reason. Interestingly, Information content based measure with support of corpus statistics has very similar performance with the other two types of measure. This indicates that the corpus statistics is fit with ontology structure of MeSH and does not improve path based measure. The measure of Mao et al. achieves the best result in both indexing schemes as shown in table 3 & 4. The reason might be that it is the only measure that utilizes the number of descendants information of compared terms. Judging from the overall performance, Wu et al., Li et al., Mao et al., Resink and the two feature based measures have a rather more stable performance than that of others. Moreover, for almost all the cases as shown in table 3, the four evaluation metrics are consistent with each other except that the score of *F-measure* and *Purity* of Wu et al. and Li et al is slightly better than baseline concept without re-weighting while *NMI* score of them is slightly worse.

From table 3 & 4, it's easily seen that the overall performance of descriptor scheme is very consistent with and slightly better than that of entry term scheme, which shows that making a document vector more precise by mapping synonym entry terms to one descriptor terms has positive effects on document clustering. It's also noted that both indexing schemes without term re-weighting have competitive performance to those with term re-weighting. It shows that term re-weighting as a method of integrating domain ontology to clustering might not be an effective approach, especially when the documents are short of terms, because when all these terms are very important core terms for the documents, ignoring the effects of some of them by re-weighting can cause serious information loss. This is in contrast to the experiment results in general domain where document length is relatively longer [3].

It's obvious that word indexing scheme achieves the best clustering result although it's not statistically significant (The word scheme experimental result is listed in both table 3 & 4 for convenience of reader). However, this does not mean indexing medical documents using MeSH entry term or MeSH descriptor is a bad scheme. In other words, it does not mean domain knowledge is not good. First, while keeping

Table 4. Clustering results of MeSH descriptor terms scheme; each measure is followed by the threshold of similarity value (in parenthesis) that helps achieve the best results

Type of Measure	Similarity Measure	Entropy	F-Score	Purity	NMI
Path based	Wu & Palmer (0.8)	0.361	0.789	0.883	0.771
	Li et al. (0.7)	0.339	0.756	0.877	0.780
	Leacock (0.2)	0.485	0.749	0.907	0.720
	Mao et al. (0.8)	0.259	0.831	0.907	0.814
Information Content	Resink (0.0)	0.346	0.815	0.890	0.777
	Jiang(0.1)	0.529	0.703	0.809	0.696
	Lin (0.9)	0.683	0.582	0.775	0.631
Feature based	Basic Feature (0.8)	0.385	0.778	0.873	0.760
	Knappe (0.8)	0.375	0.784	0.866	0.765
MeSH descriptor	None	0.341	0.772	0.867	0.776
Word	None	0.245	0.755	0.908	0.820

competitive clustering results, not only the dimension of clustering space but also the computational cost is dramatically reduced especially when handling large datasets. Second, existing ontologies are under growing, they are still not enough for many text mining applications. For example, there are only 28533 unique entry terms for the time of writing. Third, there is also limitation of term extraction. So far, existing approaches usually use “exact match” to map abstract terms to entry terms and can not judge by the sense the phrase. This will cause serious information loss. For example, when representing document as entry terms, the average document length is 14, while the length of the word representation is 81. Finally, if taking advantage of both medical concept representation and informative word representation, the results of text mining application can be more convincing.

5 Conclusion

In this paper, we evaluate the effects of 9 semantic similarity measures with a term re-weighting method on document clustering of PubMed document sets. The k-means clustering experiment shows that term re-weighting as a method of integrating domain knowledge has some positive effects on medical document clustering, but might not be significant. In detail, we obtain following interesting findings from the experiment by comparing 8 semantic similarity measures three types: path based, information content based and feature based measure with two indexing schemes—MeSH entry term and MeSH descriptor: (1) Descriptor scheme is relatively more effective on clustering than entry term scheme because synonym problem is well handled. (2) There is no a certain type of measures is significantly better than others since most of these measures consider only the path between compared concepts and their depth information within the ontology. (3) Information content based measure using corpus statistics, as well as ontology structure, does not necessarily improve the clustering result when corpus statistics is very consistent with ontology structure (4) As the only similarity measure using the number of descendants information of compared concepts, the measure of Mao et al. has the best clustering result compared to other

similarity measure. (5) Similarity measure that is not scored between 1 and 0 needs to be normalized, otherwise they will aggravate term weight much more aggressively. (6) Over all, term re-weighting achieves similar clustering result with that without term re-weighting. Some of them outperform the baseline, some of them don't and neither of them is very significant, which may indicate that term re-weighting might not be an effective approach when documents are short of terms because when most of these terms are distinguish core terms for a document, ignoring some of them by re-weighting will cause serious information loss. (7) The performance of MeSH term based schemes are slightly worse than that of word based scheme, which can be resulted from the limitation of domain ontology and limitation of term extraction and sense disambiguation. However, while keeping competitive results, indexing using domain ontology dramatically reduces the dimension of clustering space and computational complexity. Furthermore, this finding indicates that there should be an approach taking advantage of both medical concept representation and informative word representation.

In our future work, we may consider other biomedical ontology such as Medical Language System (UMLS) and also expand this comparative study to some public domain.

Acknowledgments. This work is supported in part by NSF Career grant (NSF IIS 0448023), NSF CCF 0514679, PA Dept of Health Tobacco Settlement Formula Grant (No. 240205 and No. 240196), and PA Dept of Health Grant (No. 239667).

References

1. Banerjee, A. and Ghosh, J. Frequency sensitive competitive learning for clustering on high-dimensional hyperspheres. Proc. IEEE Int. Joint Conference on Neural Networks, pp. 1590-1595.
2. Hotho, A., Staab, S. and Stumme, G., "Wordnet improves text document clustering," in Proc. of the Semantic Web Workshop at 26th Annual International ACM SIGIR Conference, Toronto, Canada, 2003.
3. Jing, J., Zhou, L., Ng, M. K. and Huang, Z., "Ontology-based distance measure for text clustering," in Proc. of SIAM SDM workshop on text mining, Bethesda, Maryland, USA, 2006.
4. Jiang, J.J. and Conrath, D.W., Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy. In Proceedings of the International Conference on Research in Computational Linguistic, Taiwan, 1998.
5. Knappe, R., Bulskov, H. and Andreasen, T.: Perspectives on Ontology-based Querying, International Journal of Intelligent Systems, 2004.
6. Larsen, B. and Aone, C. Fast and effective text mining using linear-time document clustering, KDD-99, San Diego, California, 1999, 16-22.
7. Leacock, C. and Chodorow, M., Filling in a sparse training space for word sense identification. ms., March 1994.
8. Li, Y., Zuhair, A.B., and McLean, D.. An Approach for Measuring Semantic Similarity between Words Using Multiple Information Sources. IEEE Transactions on Knowledge and Data Engineering, 15(4):871-882, July/August 2003.

9. Lin, D., Principle-Based Parsing Without Overgeneration. In Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL'93), pages 112-120, Columbus, Ohio, 1993.
10. Mao, W. and Chu, W. W., "Free text medical document retrieval via phrased-based vector space model," in Proc. of AMIA'02, San Antonio, TX, 2002.
11. Pedersen, T., Pakhomov,S., Patwardhan,S. and Chute, C., Measures of semantic similarity and relatedness in the biomedical domain. *Journal of Biomedical Informatics*, In Press, Corrected Proof, June 2006.
12. Resnik, O., Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity and Natural Language. *Journal of Artificial Intelligence Research*, 11:95-130, 1999.
13. Steinbach, M., Karypis, G., and Kumar, V. A Comparison of document clustering techniques. Technical Report #00-034, Department of Computer Science and Engineering, University of Minnesota, 2000.
14. Varelas, G., Voutsakis, E., Raftopoulou, P., Petrakis, E. G., and Milios, E. E. 2005. Semantic similarity methods in wordNet and their application to information retrieval on the web. *WIDM '05*. ACM Press, New York, NY, 10-16.
15. Wu, Z. and Palmer, M.. Verb Semantics and Lexical Selection. In *Proceedings of the 32nd Annual Meeting of the Associations for Computational Linguistics (ACL'94)*, pp133-138, Las Cruces, New Mexico, 1994.
16. Yoo I., Hu X., Song I-Y., Integration of Semantic-based Bipartite Graph Representation and Mutual Refinement Strategy for Biomedical Literature Clustering, in the Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD 2006), pp 791-796
17. Zhang X., Zhou X., Hu X., Semantic Smoothing for Model-based Document Clustering, accepted in the 2006 IEEE International Conference on Data Mining (ICDM'06).
18. Zhou, X., Zhang, X., and Hu, X., The Dragon Toolkit, Data Mining & Bioinformatics Lab, iSchool at Drexel University, <http://www.ischool.drexel.edu/dmbio/dragontool>
19. Zhao, Y. and Karypis, G. Criterion functions for document clustering: experiments and analysis, Technical Report, Department of Computer Science, University of Minnesota, 2001.

An Adaptive and Efficient Unsupervised Shot Clustering Algorithm for Sports Video

Jia Liao¹, Guoren Wang¹, Bo Zhang¹, Xiaofang Zhou², and Ge Yu¹

¹ College of Information Science & Engineering,
Northeastern University, Shenyang, China

² School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane, Australia
liaojia_email@yahoo.com.cn, wanggr@mail.neu.edu.cn

Abstract. Due to its tremendous commercial potential, sports video has become a popular research topic nowadays. As the bridge of low-level features and high-level semantic contents, automatic shot clustering is an important issue in the field of sports video content analysis. In previous work, many clustering approaches need some professional knowledge of videos, some experimental parameters, or some thresholds to obtain good clustering results. In this article, we present a new efficient shot clustering algorithm for sports video which is generic and does not need any prior domain knowledge. The novel algorithm, which is called Valid Dimension Clustering(VDC), performs in an unsupervised manner. For the high-dimensional feature vectors of video shots, a new dimensionality reduction approach is proposed first, which takes advantage of the available dimension histogram to get "valid dimensions" as a good approximation of the intrinsic characteristics of data. Then the clustering algorithm performs on valid dimensions one by one to furthest utilize the intrinsic characteristics of each valid dimension. The iterations of merging and splitting of similar shots on each valid dimension are repeated until the novel stop criterion which is designed inheriting the theory of Fisher Discriminant Analysis is satisfied. At last, we apply our algorithm on real video data in our extensive experiments, the results show that VDC has excellent performance and outperforms other clustering algorithms.

1 Introduction

In the past a few years, more and more sports videos are being produced, distributed and made available all over the world, thus, as an important video domain, sports video has been widely studied due to its tremendous commercial potential.

Different from other categories of video such as news, movie, sitcom, etc., sports video has its own special characteristics [1]. A sports game usually occurs at a specific field and always has its own well-defined content structures and domain-specific rules. In addition, sports video is usually taken by some fixed cameras which have some fixed motions in the play field, and that results in some

recurrent distinctive scenes throughout the video. For example, in a basketball game video, there are always four dominant scenes including play field, close-up of players, distant-view of players and audiences. To well understand sports video, how to take full advantage of dominant scenes is important. Video shot which comprises a sequence of interrelated consecutive frames taken continuously by a single camera represents a continuous action in time and space, and it is the basic unit of video scene. Since video shots of a scene are usually similar, merging similar shots into clusters becomes useful for the analysis of dominant scenes and even for the high-level contents of videos.

For shot clustering, some conventional algorithms such as k -means clustering and hierarchical clustering have been exploited recently [2] [3]. These methods, however, all require some prior domain knowledge to obtain good clustering results. Apart from this, these existing clustering algorithms all have their intrinsic limitations to process high-dimensional data.

In this article, we put forward a novel shot clustering algorithm for sports video, and the main contributions of our work is listed as follows. First, a new dimensionality reduction approach is proposed. By applying available dimension histogram(ADH), only valid dimensions are extracted to achieve the goal of dimensionality reduction. Second, in the subspace of valid dimensions, according to the different essentialities of them, our clustering algorithm performs on valid dimensions one by one to get more encouraging clustering results. Third, a novel stop criterion for the iterative merging and splitting procedures of each valid dimension is designed based on the theory of Fisher Discriminant Analysis.

The rest of this paper is structured as follows. Section 2 will introduce the novel dimensionality reduction approach. The details of our shot clustering algorithm will be discussed in section 3. In section 4, the performance study will be described. Section 5 will give some related work while section 6 will conclude the paper and suggest the future work.

2 Dimensionality Reduction

In this section, we will discuss our dimensionality reduction approach in detail. The valid dimension is introduced first, then how to extract valid dimensions by available dimension histogram(ADH) to achieve the goal of dimensionality reduction is proposed.

2.1 Valid Dimension

For high-dimensional data, not all dimensions are useful for different applications. In many applications, such as clustering, indexing, information retrieval, only some special dimensions are needed. Figure 1 shows an example of clustering. According to the distribution of the data set in (a), if we want to partition the points into three clusters, the clustering results can be easily found out by computing the distances among the points in the feature space of dimension d_1 and d_2 . But in fact, we have no use to take both of the two dimensions into

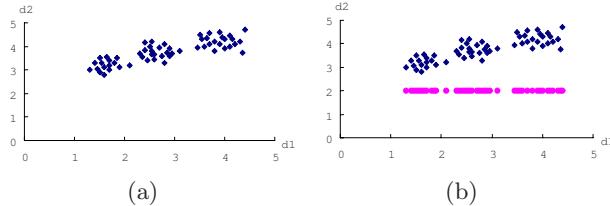


Fig. 1. An example of valid dimensions for clustering

account, only dimension d_1 is enough. (b) shows that the clustering results obtained by only considering dimension d_1 are the same as the clustering results in (a). Therefore, dimension d_1 is contributing for clustering, and it is a valid dimension of the data set.

Valid dimensions are the dimensions which can maximally represent the intrinsic characteristics of data set. For the data set in Figure 1, the standard deviations of dimension d_1 and d_2 are 0.95 and 0.48 respectively. The reason why dimension d_1 is valid for clustering is that its standard deviation is larger and it can represent the distribution of the data set. Standard deviation of a data set is a measure of how spread out it is [11]. The larger the standard deviation is, the more spread out from the mean the data set is. The data set which is more spread out is more sensitive in clustering, therefore, the dimension whose standard deviation is larger is more helpful for clustering.

The dimensionality reduction approach in this paper is to extract the valid dimensions for our clustering algorithm. In next subsection, we will discuss the extraction rule for valid dimensions.

2.2 Extraction Rule for Valid Dimensions

Sports videos have their own intrinsic characteristics. The variety of the backgrounds of sports video is not obvious. By carefully observing the high-dimensional feature vectors of video shots, it can be easily found that a mass of dimensions' values are all zero, especially in the color features. In other words, these dimensions are useless for computation, and the dimensions whose values are non-zero are called available dimensions in our paper. Table 1 gives an example of the ratio of available dimensions over the total dimensions of different categories of sports. It illustrates that the ratios of available dimensions are about 50%, thus, extracting the available dimensions is the first step of our dimensionality reduction approach.

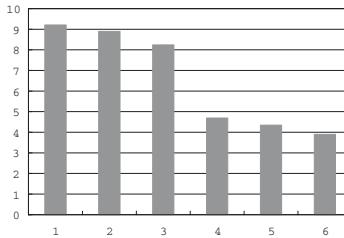
Let D_m be the subspace of data set with m available dimensions. For the values of the j th dimension of D_m , $S_i[j]$ denotes the value of shot S_i , $\sigma_{S[j]}$ denotes the standard deviation of j th available dimension of D_m , where $1 \leq j \leq m$. Standard deviation of each available dimension indicates its essentiality for clustering. Larger $\sigma_{S[j]}$ illustrates that the data in j th available dimension are more spread out and more advantageous for clustering.

Table 1. Ratios of available dimensions of different sports videos

Video data	Total dimension	Available dimension	Ratio
Basketball	512	314	61.3%
Table tennis	512	305	59.6%
Football	512	253	49.4%

Definition 1. *valid dimensions.* Given a threshold value ε , the available dimensions whose standard deviations are equal to or greater than ε are called valid dimensions.

Definition 2. *available dimension histogram(ADH).* The available dimension histogram of D_m represents the distribution of m available dimensions' standard deviations($\sigma_{S[j]}$), in which, the x -axis represents the rank of available dimensions and the y -axis represents their corresponding $\sigma_{S[j]}$. ADH displays the descending trend of $\sigma_{S[j]}$ values. The following Figure 2 gives an example of ADH.

**Fig. 2.** Example of available dimension histogram(ADH)

In order to extract valid dimensions which can maximally represent the distribution of data for clustering, an heuristic method on ADH is applied for determining the value of ε . Let $r[i]$ denote the rank of available dimensions in D_m corresponding to ADH. Then $\varepsilon = \sigma_{r[k]}$, only if $\sigma_{r[k]} - \sigma_{r[k+1]} = \max(\sigma_{r[i]} - \sigma_{r[i+1]}, 1 \leq i \leq m-1)$. ε is the standard deviation of available dimension $r[k]$ whose difference to that of $r[k+1]$ is largest in D_m . That means ε is the largest plunge occurs in VDH. Referring to Figure 2, the largest drop of ADH occurs from $r[3]$ to $r[4]$, i.e., $\varepsilon = \sigma_{r[3]}$, and the available dimensions which correspond to $r[1]$, $r[2]$, and $r[3]$ are the valid dimensions. Intuitively, such extraction rule guarantees the most significant available dimensions are extracted as valid dimensions for our clustering.

3 Unsupervised Shot Clustering Algorithm

In this section, we will provide an efficient shot clustering algorithm called valid dimension clustering(VDC) in detail.

3.1 Algorithm Description of Valid Dimension Clustering

A video shot S_i can be represented as: $S_i = \{x_1^i, x_2^i, \dots, x_n^i\}$, where x_p^i is the p th dimension of the n -dimensional feature vector S_i . Let D_f be the subspace of valid dimensions, where f is the number of valid dimensions which are obtained by our dimensionality reduction approach.

Valid dimension clustering(VDC) is an unsupervised clustering algorithm which performs on D_f one by one, that's because different valid dimensions have their own different essentialities for clustering. After ranking the standard deviations of valid dimensions in descending order, we first take the valid dimension whose standard deviation is the largest as the beginning of the algorithm, then the following valid dimensions are taken into account in order.

For the first valid dimension, each shot is first initialized as one cluster, then the iterations of merging similar shots into one cluster are repeated until the stop criterion is satisfied. For other valid dimension d_i , the clustering results of valid dimension d_{i-1} (the prior dimension of d_i according to the rank of valid dimensions) should be set as the initial clustering status of d_i , then the same merging procedures perform on each initial cluster of d_i until all initial clusters have been processed. After finishing valid dimension d_i , the algorithm will turn to d_{i+1} . The final clustering results will be returned when all f valid dimensions are processed. It is obvious that for each valid dimension, only merging procedures are performed, but for two consecutive valid dimensions d_{i-1} and d_i , the processing of d_i is splitting procedures for d_{i-1} . Thus, VDC comprises both merging and splitting procedures.



Fig. 3. Different clustering results for table tennis

The reason why VDC performs on valid dimensions one by one is explained by Figure 3. (a) gives the clustering results of VDC, i.e., valid dimensions are taken into account one by one. While (b) shows the results of the algorithm which all valid dimensions are taken into account once. Obviously, the results in (a) are better than (b). Originally, all the six shots are play field shots, but (b) partitions them into two clusters as different positions of the play table. The reason is that when we consider all valid dimensions together, all valid dimensions are treated fairly, the different essentialities of different valid dimensions have not been distinguished.

3.2 Stop Criterion of Valid Dimension Clustering

The stop criterion for the iterations is the most critical technique of unsupervised clustering algorithm. It directly determines the results of clustering. In the paper, we devise a novel stop criterion which uses Fisher Discriminant Analysis for reference.

Fisher Discriminant Analysis is a widely used multivariate statistical technique [12]. The discrimination function can be used as a well-defined rule in order to optimally assign a new observation into the labeled class. Consider k populations G_1, G_2, \dots, G_k , each with p -variate distribution which is denoted as (x_1, x_2, \dots, x_p) . Fisher suggested finding a linear combination of multivariate observations (x_1, x_2, \dots, x_p) to create univariate observation $u(x)$ such that $u(x)$ can separate the different samples of different populations as much as possible. Fisher discriminant function can be written as:

$$u(x) = \alpha^T x = \alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_p x_p \quad (1)$$

Let SSE and SSG denote the total within-class divergence and total between-class divergence of each data sample. The α which maximize the criterion $F(\alpha)$ is used in the Fisher discriminant function, the formula (II). $F(\alpha)$ is represented as below:

$$F(\alpha) = \frac{SSG}{SSE} = \frac{\alpha^T B \alpha}{\alpha^T E \alpha} \quad (2)$$

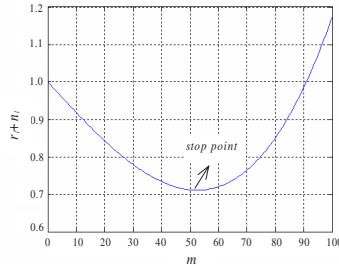
For our shot clustering algorithm, we are only interested in the concepts of within-class divergence and between-class divergence. For clustering, the intra-distance within a cluster and the inter-distance among different clusters can be mapped into the concepts of within-class divergence and between-class divergence respectively. The clustering results in which the intra-distance of each cluster is smallest and the inter-distances among different clusters are largest are the encouraging results. That indicates the data set is separated optimally.

Let r_l denote the ratio of the intra-distance of one cluster over the inter-distances among clusters when the number of clusters is N_l , and the best clustering result we want is the one with smallest value of r_l . The value of r_l can be calculated by the formula below:

$$r_l = \frac{\sum_{c=0}^{N_l} d_w^c}{d_t} = \frac{\sum_{c=0}^{N_l} \sum_{i=0}^{m_c} |S_i^c - S_{mean}^c|}{\sum_{j=0}^N |S_j - S_{mean}|} \quad (3)$$

where d_t is the initial distance among clusters, d_w^c is the intra-cluster distance of cluster c . N is the initial number of clusters at the beginning, while m_c is the number of shots in cluster c . $|\bullet|$ denotes the Manhattan distance. S_i^c and S_{mean}^c represent the i th shot and the mean vector of cluster c respectively, while S_j and S_{mean} are used for denoting the same concept of the initial clusters.

Apart from r_l , another important factor n_l is considered in our algorithm too, which is the statistic information of the number of clusters. Let $n_l = N_l/N$ be

**Fig. 4.** Relation curve of $r_l + n_l$ and m **Algorithm 1.** VDC()**Input:** ranking array of valid dimensions $r[k]$; cluster structures CR**Output:** clustering results

```

1: for  $d_n=1$  to  $k$  do
2:    $ptr=GetHead(CR)$ 
3:   while  $ptr \neq \text{NULL}$  do
4:      $S=ODC(ptr, d_n)$  // S denotes the splitting results
5:      $InsertFront(CR, S)$ 
6:      $ptr=GetNext(ptr)$ 
7:    $d_n++$ 
8: end while
9: end for

```

Function ODC(CR, d_n)initialize each shot S_i as one cluster C_i Let $r_l^{(1)} = 0, n_l^{(1)} = 1$, calculate $dist(C_i, C_j)_{d_n}, 1 \leq i, j \leq N_l$ execute **MergeCluster()**WHILE $r_l^{(1)} + n_l^{(1)} > r_l^{(2)} + n_l^{(2)} \cap N_l > 1$ $r_l^{(1)} = r_l^{(2)}, n_l^{(1)} = n_l^{(2)}$ execute **MergeCluster()**

ENDWHILE

add the clustering results to CR

end Function**Function MergeCluster()**

merge two most similar shots into one cluster

calculate $r_l^{(2)}, n_l^{(2)}$ and $r_l^{(2)} + n_l^{(2)}$ **end Function**

the ratio of the cluster number N_l over the initial total number of shots N . In order to maximally approximate the real cluster number which is a small value, the smaller the value of n_l is, the better the clustering result is.

At the beginning of the clustering algorithm, each shot is initialized as one cluster, the value of r_l is 0, and the value of n_l is 1. Then as the merging proceeds, the value of r_l is increasing while n_l is descending. When all the shots are merged

into one cluster, the value of r_l reaches 1, and n_l reaches its smallest value. Since the encouraging clustering results should have both smaller r_l and n_l , we choose $\min(r_l + n_l)$ as the stop criterion of our algorithm. When $r_l + n_l$ reaches its smallest value, the iterations of merging stop. For example, the relation curve of the value of $r_l + n_l$ and the times of iterations m for one valid dimension of football is shown in Figure 4. The inflection of the curve which corresponds to the smallest value of $r_l + n_l$ is the stop point of the iterations.

After presenting the stop criterion for iterations of our clustering algorithm, the detailed algorithm description of VDC is described in Algorithm 1.

4 Performance Study

In this section, we will report our extensive performance study on large real video data, and the comparison results with other two clustering algorithms.

4.1 Experiments Set Up

Our data set consists of about 4.5 hours' long video data which includes three categories of sports video captured from TV stations. The formats of them are all MpGs with the frame extraction rate is 25fps, and each frame is 320*240 pixels. After shot boundaries detecting, each shot is represented by feature vectors in four high-dimensional degrees: 288-D, 320-D, 384-D and 512-D which all compose HSV color feature and motion feature in P -frames of it for experiments.

Table 2. Data set statistics

Video	Length	Total shots	Cluster of shots(shot number)
Basketball(B)	1:07:25	390	C1:play field(145);C2:close-up of player(67) C3:distant view of player(150);C4:audience(28)
Table tennis(T)	1:22:32	634	C1:play field(220);C2:close-up of player(335) C3:distant view of player(47);C4:audience(32)
Football(F)	1:35:51	630	C1:play field(182);C2:close-up of player(275) C3:distant view of player(93);C4:shooting(56) C5:audience(24)

In order to detect the efficiency of our algorithm, we manually identify each shot into different clusters beforehand according to video grammar. The total number of shots in our data set is 1654, and the detailed information is listed in Table 2. Two common used measurements which are $Precision(P)$ and $Recall(R)$ are used to evaluate the performance of our algorithm. And all the experiments were done with Intel Pentium D820 processor(2.8GHz CPU's with 1GB RAM).

4.2 Effectiveness of Valid Dimension Clustering(VDC)

In order to show the excellent performance of our algorithm, other two clustering algorithms are applied in our experiment as comparisons. One is called FDC

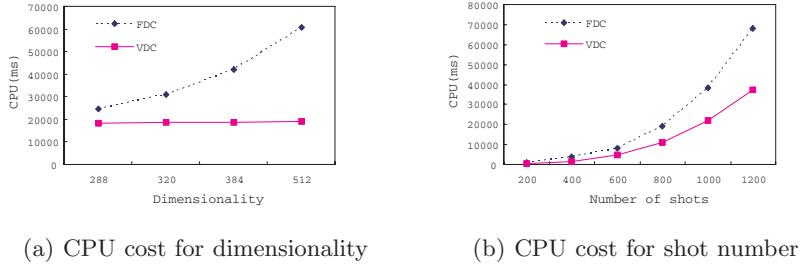


Fig. 5. Effect of dimensionality reduction

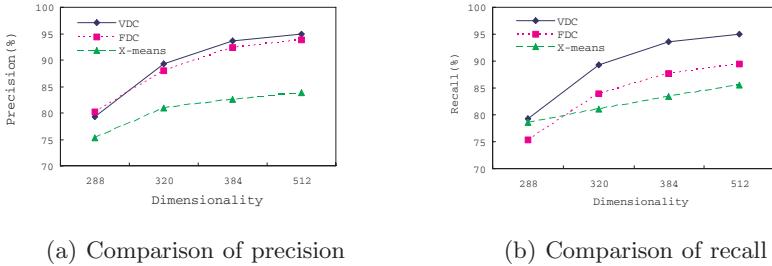


Fig. 6. Effect of dimensionality

which applies our stop criterion for merging iterations but performs on the whole high-dimensional feature space without dimensionality reduction. The other is called *X*-means [6] which is a reformative algorithm of *k*-means.

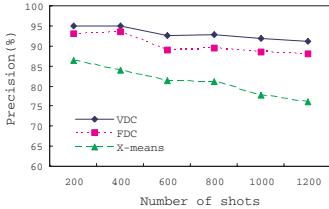
Efficiency of Dimensionality Reduction. First, we will test the efficiency of our dimensionality reduction approach which applies the available dimension histogram(ADH).

Figure 5 depicts the CPU time improvement achieved by VDC over FDC on the data sets with different dimensionality and different data size.

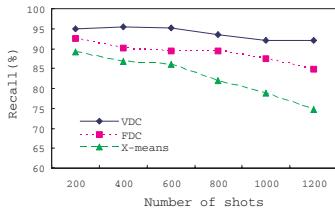
This experiment confirms that dimensionality reduction is outstanding and necessary for clustering. When the dimensionality and the size of data are increasing, the CPU time of VDC and FDC are all increasing. But it can be easily witnessed that the increasing rates of VDC are much slower than FDC, especially in (a). Obviously, dimensionality reduction plays an important role. By dimensionality reduction, only valid dimensions are considered in clustering, thus the algorithm is sped up.

Performance Comparison. In this experiment, we compare VDC with other two shot clustering algorithms. We test the effect of different dimensional feature spaces and different categories of sports video.

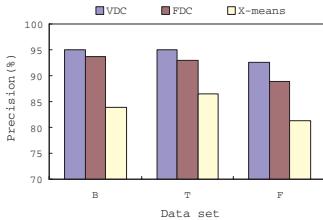
Figure 6 shows the *Precision* and *Recall* of different clustering algorithms as the dimensionality of Basketball video shots increases. When the dimensionality is increasing, *Precision* and *Recall* of all the three algorithms are improved



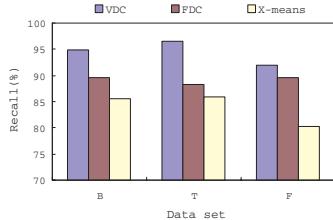
(a) Comparison of precision



(b) Comparison of recall

Fig. 7. Effect of data size

(a) Comparison of precision

**Fig. 8.** Effect of different categories of sports

because more information are extracted in higher dimensional feature space. Although the *Precision* of VDC is only slightly larger than that of FDC, the *Recall* of FDC is much smaller. VDC performs best and outperforms *X*-means by nearly 10%.

Figure 7 shows their *Precision* and *Recall* when the number of shots increases. For VDC and FDC, following with the increasing number of shots, *Precision* and *Recall* only change a little, and the average values of them are much larger than those of *X*-means. Because of the disadvantage on deciding data centers and number of clusters, the performance of *X*-means is dissatisfactory comparing with others.

The performances of clustering algorithms varies from different categories of sports due to some factors such as positions of cameras, motions of players, and common screens. Figure 8 shows the distinct performances on the three categories of sports. Obviously, for table tennis, all the three algorithms perform best. The main reason is that the play field of table tennis is small, the number of common screens and the motions of cameras are almost fixed for table tennis, thus it's easier to achieve better results for clustering algorithm.

Table 3 illustrates the detailed experimental results of our data sets. For each cluster which we pre-divided manually, we evaluate the performances of the algorithms by three measurements: number of clusters C_n , *Precision*(P), and *Recall*(R). In the table, it can be easily found that C_n of VDC is smaller than other two algorithms and is closer to the manually divided cluster number for

Table 3. Data set statistics

Video	Results of VDC			Results of FDC			Results of X -means			
	C_n	P(%)	R(%)	C_n	P(%)	R(%)	C_n	P(%)	R(%)	
Basketball:	C1	2	97.17	93.54	5	100.0	89.66	7	83.77	85.28
	C2	3	97.91	98.85	4	92.62	90.84	5	84.32	87.22
	C3	2	91.26	95.08	4	88.43	87.72	6	82.81	85.77
	C4	1	96.82	91.14	3	89.38	95.10	3	89.13	82.78
Table tennis:	C1	2	95.65	100.0	4	100.0	95.45	6	86.45	83.52
	C2	2	96.33	95.38	5	87.95	84.86	8	86.74	88.93
	C3	3	85.97	91.67	3	90.64	85.17	4	87.09	77.47
	C4	1	91.42	90.88	2	100.0	81.25	2	81.98	84.76
Football:	C1	3	91.41	90.41	5	82.93	94.44	6	75.68	85.11
	C2	2	95.42	92.55	4	92.30	87.48	6	85.49	77.42
	C3	2	90.63	89.15	5	90.06	90.18	4	87.53	84.26
	C4	2	85.74	100.0	2	91.25	84.79	3	67.68	71.95
	C5	1	91.00	88.10	1	87.37	83.66	2	81.06	79.82

different sports. That's an important advantage of our algorithm which means the clustering results of VDC are more credible. Set basketball video as an example, VDC obtains 8 clusters, FDC and X -means obtain 16 and 21 respectively, while the manual cluster number is only 4. Since VDC considers most contributing characteristics of data sets which correspond to valid dimensions and performs on them one by one, it achieves most reasonable clustering results. In addition, the *Precision* and *Recall* of VDC are better than those of FDC and X -means. Among the results of VDC, there are 11 clusters whose *Recall* are above 90%, while only 5 clusters whose *Recall* are above 90% in the results of FDC. *Precision* and *Recall* of X -means are both smallest and the performance is disatisfactory. And the average value of *Precision* and *Recall* of VDC are 92.82% and 93.59% respectively. As a whole, the performance of VDC is desirable.

5 Related Work

Clustering techniques are intended to group data with similar attributes into clusters that exhibit certain high-level semantics. In previous work, most of the clustering algorithms which are used in the field of video data require some parameters to obtain good results. In [4], a shot cluster is split when its variance is above a pre-specified threshold, and two shot clusters are merged into one when the distance between their centers is below another pre-defined threshold. In [5], the number and initial centers of shot clusters are required by k -means clustering algorithm. But it is well known that the estimation of correct cluster number and the decision of good cluster centroids had been longstanding problems in cluster analysis. [6] reported a reformative algorithm of k -means which is called X -means, and it applied Bayesian information criterion to estimate the number of clusters.

For high-dimensional data, the performances of most existing clustering algorithms degrade rapidly [8]. Thus, to minimize the effect of "dimensionality curse" before processing high-dimensional data becomes more important. The technique which using Principle Component Analysis to reduce the dimensionality of data works well when the data set is global correlated [8]. [7] proposes a new approach which dimensionality reduction procedures are dynamic and can be adaptively adjusted and integrated with the clustering processing. Especially for video data, there are also a few works to reduce dimensionality of video data. [9] introduces a new one dimensional transformation technique which rotates and shifts the original axis system using PCA. To speed up the shot clustering process and minimize the space requirement, [10] applies both PCA and LDA techniques to reduce the dimension of feature vectors for the domain scenes clustering algorithm.

6 Conclusions and Future Work

In this paper, we introduce a novel unsupervised shot clustering algorithm for sports video called valid dimension clustering(VDC). We first apply a new dimensionality reduction approach to get "valid dimensions". Then in the subspace of valid dimensions, VDC performs on valid dimensions one by one. After that, the iterations of merging and splitting repeat until the novel stop criterion which is designed inheriting the theory of Fisher Discriminant Analysis is satisfied without any parameters. At last, our extensive experiments on real sports video prove the effectiveness and efficiency of our proposals.

For our future work, we first plan to further investigate the effectiveness and efficiency of our algorithm on different kinds of videos. Second, cross-media information, such as audio, text information, and image will be taken into account to improve the performance of clustering. Third, more efficient dimensionality reduction approaches for video data will be also considered in the future.

Acknowledgments. This work is partially supported by National Natural Science Foundation of China under grant No.60573089 and 60273079, and supported by National Basic Research Program of China(973) under Grant No. 2006CB303103.

References

1. M. Bertini, A.D. Bimbo, R. Cucchiara, and A. Prati. Semantic video adaptation based on automatic annotation of sport videos. In *Proc. of the 6th ACM SIGMM Int. workshop on Multimedia Information Retrieval*, pp.291-298, 2004.
2. A. Hanjalic and H. Zhang. An integrated scheme for automated video abstraction based on unsupervised cluster-validity analysis. In *IEEE Transactions on Circuits and Systems for Video Technology*, Vol.9(8), pp.1280-1289, 1999.
3. M. Yeung, B. L. Yeo, and B. Liu. Extracting story units from long programs for video browsing and navigation. In *Proc. IEEE Conf. on Multimedia Computing and System*, pp.296-305, 1996.

4. X. Q. Zhu, J. P. Fan, AK.Elmagarmid, *et al.* Hierarchical video content description and summarization using unified semantic and visual similarity. In *Journal of Multimedia Systems*, Vol.9(1), pp.1432-1882, 2003.
5. J. Pena, J. Lozano, and P. Larraaga. An empirical comparison of four initialization methods for the k-means algorithm. In *Pattern Recognition Letters*, Vol.20, pp.1027-1040, 1999.
6. D. Pelleg, A. W. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proc. of the 17th Int. Conf. on Machine Learning*, pp.727-734, 2000.
7. C. Ding, X.F. He, H.Y. Zha, *et al.* Adaptive dimension reduction for clustering high dimensional data. In *Proc. of ICDM*, pp.147-155, 2002.
8. R. Agarwal, J. Gehrke, D. Gunopulos, *et al.* Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. of Int. Conf. on Management of Data*, pp.94-105, 1998.
9. H. T. Shen, B. C. Ooi, and X. F. Zhou. Towards effective indexing for very large video sequence database. In *Proc. of Int. Conf. on Management of Data*, pp.730-741, 2005.
10. H. Lu and Y. P. Tan. Unsupervised clustering of dominant scenes in sports video. In *Pattern Recognition Letters*, Vol.24(15), pp.2651-2662, 2003.
11. H. Yin, N. Allinson, and R. Freeman. Intelligent data engineering and automated learning. In *IDEAL*, 2002.
12. M. Sever, J. Lajovic, and B. Rajer. Robustness of the Fisher's discriminant function to skew-curved normal distribution. In *Proc. Int. Conf. of Applied Statistics*, Vol.2(2), pp.231-242, 2005.

A Robust Feature Normalization Scheme and an Optimized Clustering Method for Anomaly-Based Intrusion Detection System

Jungsuk Song¹, Hiroki Takakura², Yasuo Okabe²,
and Yongjin Kwon³

¹ Graduate School of Informatics, Kyoto University
`oaktree@net.ist.i.kyoto-u.ac.jp`

² Academic Center for Computing and Media Studies, Kyoto University
`takakura@media.kyoto-u.ac.jp, okabe@i.kyoto-u.ac.jp`

³ Information and Telecom. Eng., Hankuk Aviation University
`yjkwon@tikwon.hangkong.ac.kr`

Abstract. Intrusion detection system(IDS) has played a central role as an appliance to effectively defend our crucial computer systems or networks against attackers on the Internet. Traditional IDSs employ signature-based methods or anomaly-based methods which rely on labeled training data. However, they have several problems, for example, it consumes huge amounts of cost and time to acquire the labeled training data, and they often experienced difficulty in detecting new types of attack. In order to cope with the problems, many researchers have proposed various kinds of algorithms for several years. Although they do not require labeled data for training and have the capability to detect unforeseen attacks, they are based on the assumption that the ratio of attack to normal is extremely small. However, the assumption may not be satisfied in a realistic situation because some attacks, most notably the denial-of-service attacks, consist of a large number of simultaneous connections. Consequently if the assumption fails, the performance of the algorithm will deteriorate. In this paper, we present a new normalization and clustering method that can overcome a limitation on the attack ratio of the training data. We evaluated our method using KDD Cup 1999 data set. Evaluation results show that performance of our approach is constant irrespective of an increase in the attack ratio.

1 Introduction

In recent years, considerable attention has been given to intrusion detection on the Internet. Intrusion detection is defined as the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions. IDS is one of the systems designed to perform such intrusion detection and an integral part of any complete security package of a modern well managed network system.

Conventional IDSs employ signature-based detection, which relies on labeled training data. However, IDSs using these methods have several problems, for

example, they can only detect previously known intrusions and it consumes huge amount of cost and time to acquire the labeled training data. A survey of these methods is given in [1]. Over the past few years, several studies to solve these problems have been made on anomaly detection using unsupervised learning techniques, called unsupervised anomaly detection, which can detect previously “unseen” attacks and do not require labeled data used in training stage[23].

There are many approaches that apply unsupervised anomaly detection for intrusion detection such as clustering, one-class support vector machine(SVM), etc [4][5][6]. Although they do not require labeled data for training and have capability of detecting unforeseen attacks, they make two assumptions about the data to be trained. First, the ratio of attack to normal is extremely small[9]. Second, the attack traffic is statistically different from normal traffic[3]. It is important to note that these assumptions may not be satisfied in a realistic situation because some attacks, most notably the denial-of-service(DoS) attacks, consist of a large number of simultaneous connections, and in many cases they may be misclassified as normal because of their enormous volume. After all, if the assumption fails, performance of the algorithm will deteriorate. In this paper, we propose a new normalization and clustering method for intrusion detection. This proposed method is based on K-means clustering method[7], which is a typical clustering algorithm.

We evaluated our method over the network data from KDD Cup 1999[8], which is a very popular and widely used intrusion attack data set. Our experimental results show that performance of our approach is constant irrespective of an increase in the attack ratio, and outperforms the K-means.

The rest of the paper is organized as follows. In section 2, we give some background information about data normalization and the K-Means algorithm. In section 3 and 4, we present our normalization and clustering method in detail, respectively. In section 5, we describe the details of our experiment and present the results and their analysis. Finally, we present concluding remarks and suggestions for future study.

2 Related Work

2.1 Normalization

In many approaches that employ anomaly-based intrusion detection with unlabeled data [9][10][11][12][13], it is required to normalize the training and test data because each feature of the data instances has a different scale. For example, consider two 3-features vectors: $\{(1, 2, 100), (5, 3, 200)\}$. Under the Euclidean metric, the squared distance between the feature vectors will be $(1 - 5)^2 + (2 - 3)^2 + (100 - 200)^2 = 16 + 1 + 10,000 = 10,017$. As you see, there is a problem that the distance is dominated by the third feature.

2.2 K-Means Clustering Algorithm

Clustering is one of unsupervised learning techniques to group data instances into meaningful subclasses [4][5]. K-means[7] is one of basic methods for clustering. It partitions a set of data into k clusters through the following steps.

- Initialization: Randomly choose k instances from data set and make them initial cluster centers.
- Assignment: Assign each instance to the closest center.
- Updating: Replace every cluster’s center with the mean of its members.
- Iteration: Repeat Assignment and Updating until there is no change for each cluster, or other convergence criterion is met.

The popularity of the K-means algorithm is largely due to its low time complexity, simplicity and fast convergence. In particular, low time complexity is a significant factor for intrusion detection because it is performed over large and high-dimension network data sets. However, it has been known that the K-means algorithm has several shortcomings as follows.

First, the K-means algorithm is sensitive to the initial centers; that is, the clustering result of the K-means algorithm is dependent on the chosen initial centers. Second, high dimension of each data instance causes heavily performance deterioration of the algorithm, this is called “curse of dimensionality”. Third, the K-means algorithm is difficult to choose the number k of clusters to be created finally. Finally, the K-means algorithm just can find out the local optimum, not the global optimum. Hence, we propose a method to overcome these shortcomings of the K-means for intrusion detection. See section 4 for detail.

3 Normalization

In this section, we present a novel normalization method for preventing an increase of the anomaly ratio from decreasing the performance.

3.1 Defining of Notations

Before describing our method, it is necessary to specify about the major notations that are used in this paper:

- $S = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$: the set of data instances to be clustered
- n : the number of all data instances in the training data
- $\mathbf{x}_j = (x_{j1}, x_{j2}, \dots, x_{jd})$: a vector in real d -dimensional space, \mathbb{R}^d
- $\|\mathbf{x}\|$: the Euclidean distance of the vector(i.e. instance) \mathbf{x}
- $C = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_k\}$: the set of k cluster centers
- \mathbf{c}_j : the mean of each cluster $C_j (1 \leq j \leq k)$
- $D = \{\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_d\}$: the set of dimensions in the feature space

3.2 Methodology

Our method is basically based on [9]. In their normalization, they first calculate the average and standard deviation of every feature in the feature space. By using them, they calculate, for every feature value of each instance, how far it is away from the average of corresponding feature, and then the result divided by its standard deviation becomes the new value(i.e. normalized value) for that

Table 1. Average and change of normalized value

Ratio of 10	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Average	1.09	1.18	1.27	1.36	1.45	1.54	1.63	1.72	1.81	1.9
Normalized value of 1	-0.1	-0.14	-0.17	-0.20	-0.22	-0.25	-0.27	-0.29	-0.31	-0.33
Normalized value of 10	9.9	6.96	5.65	4.87	4.33	3.93	3.62	3.37	3.16	2.98

feature. However, there is a problem that if the ratio of the attack data increases, distinction between the normal instances and the attack instances becomes more difficult. For example, consider 100 1-feature data instances where each data instance has a value either 1(normal) or 10(attack). From Table 1, we can see that difference between the normalized value of 1 and 10 diminishes(i.e. be more difficult to distinguish) gradually with increment of the attack data instances(i.e. 10). This is because that the average value of instances is heavily affected by the number of the attack instances. As a result, it leads to performance deterioration.

Therefore, we propose a method that can maintain good performance of IDS irrespectively of normal-attack ratio. In generally, it is obvious that the number of normal traffic is lager than that of attack traffic in a real environment. It means that if a data instance is normal, there are a lot of data instances with the similar attribute value to the data instance, otherwise the number of data instances which have the similar attribute value is few. Hence, we first partition the training data into two groups: dense group and sparse group. The dense group consists of data instances whose attribute values are similar each other and frequently appear in the training data, while in case of data instances in the sparse group, similar attribute values are seldom observed in the training data. Our normalization uses the average and standard deviation from data instances only from the dense group.

Let us present the algorithm in more detail. For each dimension $\mathbf{d}_i (1 \leq i \leq d)$ where i denotes the i th dimension, we search the minimum and maximum values of the training data, and divide their difference into small equi-length partitions called *bin*, where the number of the *bins* is determined by parameter β that is supplied by user. That is,

$$\mathbf{d}_i = \mathbf{d}_{i1} \cup \mathbf{d}_{i2}, \dots, \cup \mathbf{d}_{i\beta}.$$

The algorithm repeats the dividing process for every dimension in the feature space. After the process is finished, the algorithm reads the training data again and counts the frequency of *bins*. Let n_{ib} denotes the number of data instances fall in $\mathbf{d}_{ib} (1 \leq b \leq \beta)$ and n'_{ib} denotes its ascending order; that is,

$$n'_{i1} \leq n'_{i2} \dots \leq n'_{i\beta}.$$

For all dimensions, the algorithm finds *bin* \mathbf{d}'_{il} under the following conditions.

$$\sum_{b=1}^{l-1} n'_{ib} \leq \frac{n}{\alpha} < \sum_{b=1}^l n'_{ib}$$

where α is supplied by a user and \mathbf{d}'_{il} denotes corresponding *bin* to n'_{il} . If α equals 100, for example, \mathbf{d}'_{il} is the first *bin* where summation of n'_{ib} exceeds 1% of all data instances in the training data. The algorithm then obtains following n''_{im} that represents difference between neighbor *bins*:

$$n''_{im} = n'_{im+1} - n'_{im}, \text{ for } (1 \leq m \leq l-1),$$

and we regard m of maximum n''_{im} as M : that is,

$$M = m \text{ of } \max\{n''_{im} | (1 \leq m \leq l-1)\}.$$

We then define $\{\mathbf{d}'_{i1}, \mathbf{d}'_{i2}, \dots, \mathbf{d}'_{iM}\}$ and $\{\mathbf{d}'_{iM+1}, \mathbf{d}'_{iM+2}, \dots, \mathbf{d}'_{i\beta}\}$ as “sparse region” and “dense region”, respectively. By using the sparse and dense region of each dimension, the algorithm partitions the training data into two groups: sparse group and dense group. If a data instance has at least one dimension which belongs to the sparse region, it is classified to the sparse group; otherwise it becomes a member of the dense group.

Given a set of the dense group, we calculate the average and standard deviation of every feature only using the data instances of the group. We then normalize each instance as follow:

$$\text{normalized_instnace}[j] = \frac{\text{original_instance}[j] - \text{average}[j]}{\text{standard_deviation}[j]},$$

where $[j]$ is the j th feature. In our normalization work, only numerical features were converted.

4 Clustering

In this section, we present our clustering algorithm for intrusion detection. The clustering process is basically the same as the K-means algorithm except that “Splitting” and “Merging” processes are added just after the updating process.

4.1 Selecting Initial Cluster Centers

In this process, we create k initial cluster centers. In intrusion detection, the ideal initial cluster centers are required to satisfy the condition where each instance of the training data should be classified to one of them. The training data have a lot of clusters which should be labeled either to normal or attack. Therefore, we have to find both normal and attack clusters that really exist in the training data, and then calculate the initial cluster centers using them.

In our approach, we utilize the dense and sparse groups to create k initial cluster centers. First, for representing attack clusters of the training data, the algorithm generates same groups, S_1, S_2, \dots, S_s from the sparse group. If each vector’s element of both instances belongs to the same sparse region, the algorithm treats them as the same group. Note that the number of same groups cannot exceed that of data instances in the sparse group. The algorithm also generates groups(clusters), S_{s+1}, \dots, S_k , from data instances in the dense group

by the following method. First, for one dimension, the algorithm treats all data instances in each *bin* of the dense region as member of a cluster. Note that each instance of the dense group is a member of only one *bin* in each dimension. The algorithm then repeats the process for every dimension in the feature space. As a result, the total number of generated clusters is equal to all number of the *bins* that belong to the dense region in every dimension of the feature space. Note that our algorithm does not require to determine k in advance. Finally, the algorithm calculates the mean of each cluster $S_h(1 \leq h \leq k)$ that becomes the k initial cluster centers, namely $\mathbf{c}_h(1 \leq h \leq k)$. We also denote these centers as $\mathbf{s}_h(1 \leq h \leq K)$ and K as the number of the initial cluster centers for the purpose of the labeling process(Section 4.5).

It is important to consider the following case. If the data in any particular dimension is uniformly distributed, then the dimension does not give any useful information concerning the dense and sparse region. Hence, we need to determine whether dimensions are worth investigating. We calculate $|n'_{i\beta} - n'_{iM}|(1 \leq i \leq d)$ for each dimension to extract the uniform dimensions. For the uniform dimensions, we expect them to contain data instances of almost same number with respect to each *bin*. Therefore, if a dimension satisfies $|n'_{i\beta} - n'_{iM}| < \frac{n}{\alpha}$, we regard the dimension as the uniform distribution, and our algorithm excludes the uniform distribution dimensions from the clustering process hereafter.

4.2 Allocating Data Instances

In this process, the algorithm allocates data instances to the closest cluster center. As mentioned above, we have to reduce the number of dimensions to improve performance of the proposed algorithm because as the number of dimensions in a dataset increases, distance measures become increasingly meaningless (i.e. the curse of dimensionality). Subspace clustering [14] is one of methods to reduce dimension of data instances in the clustering field. Hence, in order to reduce dimensions of data instances, we propose an allocating algorithm based on concept of subspace clustering.

Subspace clustering is the method that attempts to find meaningful clusters from different subspaces of the same dataset. In subspace clustering, the key point is that each cluster can be easily extracted from the dataset by finding appropriate subspaces (i.e., dimensions), and in such subspaces the data instances converge on the particular attribute value. Detailed description about it is given in [14].

Therefore, our allocating algorithm begins by searching “dense region dimensions” for each cluster center. We regard the searched dimensions as appropriate subspace for intrusion detection because many data instances converge on the dimensions. The algorithm then calculates the distance from data instances to each of current k cluster centers only with respect to the dense region dimensions. As different from existing researches that have to acquire the distance for all dimensions, proposed algorithm requires only dimensions which belong to the dense region of each cluster center. The algorithm then allocates each instance to the closest cluster center.

After the assignment process is finished, each cluster center has new data instances. Therefore the algorithm updates the mean of each cluster center with its new members.

4.3 Splitting and Merging Clusters

In order to overcome the K-means's two shortcomings, *the number k* and *local optimum*, we apply splitting and merging processes. There are many variants of the K-means that employ splitting and merging processes to overcome the two shortcomings such as ISODATA [16]. Although the approaches can overcome two shortcomings, they need many parameters: initial number of clusters, maximum number of iterations and so on. Therefore, we propose an algorithm that can overcome two shortcomings and does not require any additional parameters.

We first compute the values as follows:

- Δ_j : the average distance between data instances in cluster C_j and their cluster center \mathbf{c}_j
- Δ : average of $\Delta_j (1 \leq j \leq k)$
- σ_j : the standard deviation of the average distance between data instances in cluster C_j and their cluster center \mathbf{c}_j
- σ : average of $\sigma_j (1 \leq j \leq k)$

Splitting. For each cluster $C_j (1 \leq j \leq k)$, if $\Delta_j > \Delta$ and $\sigma_j > \sigma$, we search an instance that is the furthest to its center. If the distance between the instance and the center is larger than $\Delta_j + \sigma_j$, the instance should not be a member of the cluster. Thus, we create a new cluster and treat the instance as its center.

Merging. We first calculate the values as follows:

- d_{ij} : the distance between all cluster centers
- \bar{d} : the average distance of d_{ij}

That is,

$$d_{ij} \leftarrow \|\mathbf{c}_i - \mathbf{c}_j\|, \bar{d} \leftarrow \frac{\sum d_{ij}}{k(k-1)/2}, 1 \leq i < j \leq k.$$

Also, we search $d_{ij} (1 \leq i < j < k)$ whose the value is less than \bar{d} and thus the algorithm merge C_i and C_j , if two clusters satisfy two conditions as follows:

1. $\|\mathbf{c}_i - \mathbf{c}_j\| < \min(\sigma_i, \sigma_j)$
2. $|\sigma_i - \sigma_j| < (\frac{\sigma_i}{n_i} + \frac{\sigma_j}{n_j})$, n_i, n_j : the number of instances in C_i, C_j

According to the conditions, if each center of clusters C_i and C_j exists in the another cluster's region and if difference of their deviations is quite small, they should be treated as one cluster.

In this way, the value of k will be updated automatically by splitting and merging clusters. Also, local optimum problem of the K-means algorithm can be solved because if the k initial cluster centers were wrong, it can be modified by splitting and merging clusters.

4.4 Convergence Criterion

In this process, the algorithm uses the sum of the squared Euclidean distance, denoted by $E(\mathbf{c}_1, \dots, \mathbf{c}_k)$, as the convergence criterion that is most intuitive and frequently used function in partitional clustering techniques like the K-means. We can calculate $E(\mathbf{c}_1, \dots, \mathbf{c}_k)$ as follows:

$$E(\mathbf{c}_1, \dots, \mathbf{c}_k) = \sum_{j=1}^k \sum_{\mathbf{x}' \in C_j} \|\mathbf{x}' - \mathbf{c}_j\|^2$$

where \mathbf{x}' is normalized instance by the proposed method. $E(\mathbf{c}_1, \dots, \mathbf{c}_k)$ means that summation of the distance between each cluster center and its instances. If the value of $E(\mathbf{c}_1, \dots, \mathbf{c}_k)$ is larger than that of last clustering, go to the labeling process, otherwise the algorithm go back to the assignment process.

4.5 Labeling Clusters

In the existing researches, they labeled a cluster as attack if it is relatively large or is above a given threshold, otherwise, it is labeled as normal. However, such method has several limitations on labeling the clusters accurately, for example, there is not a precise criterion about relatively large or a threshold, and it can not detect an attack that causes large data instances like DoS.

Therefore, we propose a new method that does not depend on the population ratio of the cluster nor does not require a threshold to label the clusters. Although the initial cluster centers of the dense group are not proper as our desired centers, since we assume that most of those is normal, we can utilize those as a criterion for labeling. In other words, we can say that if a cluster is normal, the distance between the center $\mathbf{c}_j (1 \leq j \leq k)$ of the cluster and $\mathbf{s}_h (s+1 \leq h \leq K)$ will be small, otherwise that will be large. Thus, we first, for each cluster center $\mathbf{c}_j (1 \leq j \leq k)$, calculate the maximum distance to $\mathbf{s}_h (s+1 \leq h \leq K)$. We then calculate the average of the maximum distances. If the maximum distance from a cluster to $\mathbf{s}_h (s+1 \leq h \leq K)$ is less than the average, we label the cluster as normal. Otherwise, label as attack.

After the labeling process is finished, we calculate the distance from a data instance of the test data to $\mathbf{c}_j (1 \leq j \leq k)$ and if the label of the closest cluster from the data instance is an attack, then label the data instance as the attack. Otherwise, label the data instance as normal.

5 Experimental Results and Analysis

In order to evaluate the proposed clustering method, we tested the algorithm on a benchmark dataset, the network traffic data from KDD Cup 1999 Dataset [8]. We are interested in two indicators: the detection rate and the false positive rate. The detection rate is defined as the number of intrusion instances (correctly) detected by the system divided by the total number of intrusion instances present in the test set. The false positive rate is defined as the total number of normal instances

that were (incorrectly) classified as intrusions divided by the total number of normal instances present in the test set.

5.1 Data Set Descriptions

Training Data. In KDD Cup 1999 Dataset, the training data set consists of approximately 4.9 million data instances. Each instance consisted of 41 features of various types, and a class label that indicate either normal or one of the attack types.

Test Data. The test data consists of approximately 490,000 data instances. It contains 17 types of attack that were not present in the training data and 20 types of attack that were present in the training data.

5.2 Results

We first evaluated performance of the proposed method and K-means algorithm. For evaluation, we randomly extracted the training and test data from KDD Cup dataset. The training and test data consist of 90,373 and 65,108 instances, respectively. Around 1% of the training data is attack, and the test data has 4,515 attack instances that consist of 2,275 known attack instances(i.e. included in the training data) and 2,240 new attacks. For comparison, we obtained the false positive rate and detection rate(i.e. ROC curve[15]) of the two methods by varying α and β , and k (in case of the K-means). Parameter k was set to 3, 5, 10, 20, 50 and 100. Note that every experimental result in this paper is averaged over 10 runs of the algorithms. The comparison of ROC curves of the two methods is shown in figure 1(a). As we had expected, it can be easily seen that performance of the proposed method consistently outperforms the K-means algorithm; especially at the lower false positive. Therefore, we conclude that superior performance of the proposed clustering method results from overcoming four shortcoming of the K-means algorithm.

We also evaluated stability of the proposed method with respect to different attack ratio of the training data. As the training data for this evaluation, we prepared three different dataset where each dataset consists of 90,373 instances, and the attack ratio of those is 1%, 5% and 10%, respectively while not changing the above test data. We obtained the ROC curves of each case as shown in figure 1(b), and we obviously understand that performance of the proposed method is not influenced by the ratio of attack. It means that by applying our normalization method the average of every feature(i.e. dimension) did not move toward the anomalies. In other words, since normalized values of the attack instances are still far from the normal ones, the proposed clustering method was able to detect those excellently.

5.3 Analysis

First of all, we investigated our strategy that the proposed normalization method does not change the average of every feature(i.e. dimension) according to an

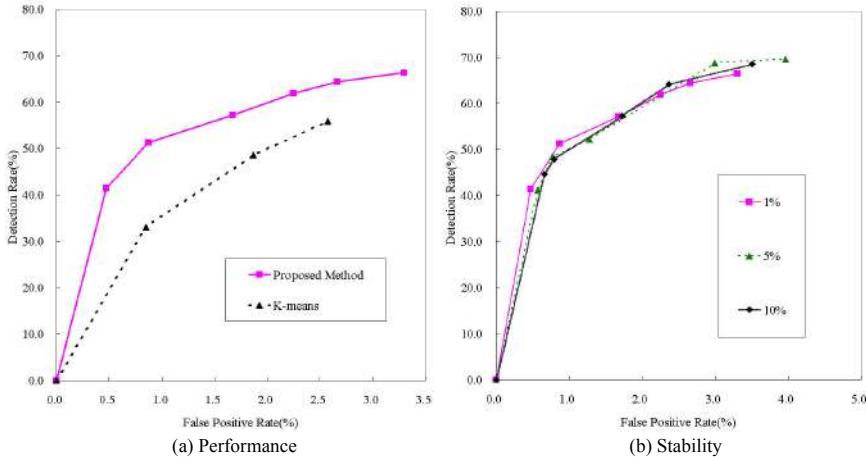


Fig. 1. ROC curves showing performance of the algorithms over KDD data set

Table 2. Average of 17 and 22 dimensions

Dimension	Proposed Method			Existing Method			Attack			Normal		
	1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%
17	7.15	7.97	7.90	7.66	12.13	13.49	50.86	60.37	62.99	7.62	9.27	8.01
22	0.95	0.96	0.96	0.95	0.94	0.93	0.82	0.78	0.77	0.95	0.95	0.95

increase in the attack ratio. Actually, in our experiments, there were many dimensions that exactly correspond with our assumption, and we take an example for 17th and 22nd dimensions as shown in Table 2. For fair comparison, we fixed $\beta = 100$ and changed $\frac{n}{\alpha} = 1\%$, $\frac{n}{\alpha} = 5\%$, $\frac{n}{\alpha} = 10\%$ for the three training data, its attack ratio is 1%, 5% and 10%, respectively. Our results show that by applying the proposed method, the average of each dimension(Proposed Method in Table 2) is not only almost constant, but also almost the same as that for only real normal data in the training data(Normal in Table 2). This invariability of the average also means that our assumption on the dense group (i.e. most of data instances included in the dense group is normal) is reliable. However, in case of the existing methods(Existing Method in Table 2), fluctuation of attack data(Attack in Table 2) induces deterioration of their average, i.e., increase of 17th dimension and decrease of 22nd one.

In addition to superiority of the proposed method in terms of performance and stability, the short detection time is an important factor for practical applying as the training time. Thus, we also measured the detection time of the proposed method. In our method, it took approximately 18 seconds to determine 65,108(around 13% of the original test data) data instances of the test data. After all, it will be take approximately 135 seconds to detect the whole test data of two weeks. It means that the proposed method enables IDSs to analyze audit data in real-time even though it requires two user defined parameters α and β .

6 Conclusion and Future Works

In this paper, we have pointed out the limitation on data normalization of anomaly-based IDS and the shortcomings of the K-means algorithm. First, we proposed a novel normalization method that can maintain constant performance of the system irrespective of the amount of attack data. Second, for improving performance of the K-means algorithm, we have proposed a clustering algorithm to overcome its shortcomings in intrusion detection.

We have evaluated the accuracy of the new approach by varying two parameters α and β . Our results showed that it achieves a higher detection rate than the K-means algorithm while maintaining a low false positive rate, and an increase in the attack ratio does not influence performance of the proposed method. Furthermore, linear time complexity and real-time detection ability of the proposed method make it feasible for intrusion detection.

For future work, we need to verify performance of the proposed clustering algorithm over real data and make a new benchmark dataset for intrusion detection, because KDD Cup 1999 dataset was generated in the virtual reality network (i.e. it can not reflect the reality) and the attacks included in it are greatly old-fashioned.

Acknowledgement. This research is partly supported by the Grant-in-Aid for Scientific Research on Priority Areas, The Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

1. C. Warrender, S. Forrest, and B. Pearlmutter, “*Detecting intrusions using system calls: alternative data models*”, In 1999 IEEE Symposium on Security and Privacy, pp. 133-145, IEEE Computer Society, 1999.
2. D. E. Denning, “*An intrusion detection model*”, IEEE Transactions on Software Engineering, SE-13:222-232, 1987.
3. H. S. Javitz and A. Valdes, “*The NIDES statistical component: description and justification*”, In Technical Report, Computer Science Laboratory, SRI International, 1993.
4. B. Everitt, S. Landau, and M. Leese, “*Cluster Analysis*”, London: Arnold, 2001.
5. A. Jain and R. Dubes, “*Algorithms for Clustering Data*”, Englewood Cliffs, NJ: Prentice-Hall, 1988.
6. Schölkopf, B., Platt, J., Shawe-Taylor, J., Smola, A., und Williamson, R., “*Estimating the support of a high-dimensional distribution*”, Neural Computation. 13(7):1443-1471, 2001.
7. MCQUEEN, J, “*Some methods for classification and analysis of multivariate observations*”, In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, pp. 281-297, 1967.
8. The third international knowledge discovery and data mining tools competition dataset KDD99-Cup <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>, 1999.

9. L. Portnoy, E. Eskin and S. Stolfo, “*Intrusion Detection with Unlabeled Data Using Clustering*”, In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security*, 2001.
10. E. Eskin, A. Arnold, M. Prerau, L. Portnoy and S. Stolfo, “*A Geometric Framework for Unsupervised Anomaly Detection : Intrusion Detection in Unlabeled Data*”, In *Applications of Data Mining in Computer Security*, 2002
11. Y. Guan, A. Ghorbani and N. Belacel, “*Y-means : A Clustering Method for Intrusion Detection*”, In *IEEE Canadian Conference on Electrical and Computer Engineering, Proceedings*, 2003.
12. Laskov, P., Schäfer, C., Kotenko, I., “*Intrusion detection in unlabeled data with quarter-sphere support vector machines*”, In: Proc. DIMVA, pp. 71-82, 2004.
13. K. Leung, and C. Leckie, “*Unsupervised Anomaly Detection in Network Intrusion Detection Using Clusters*”, In *Proceedings of Twenty-Eighth Australasian Computer Science Conference (ACSC2005)*, 2005.
14. L. Parsons, E. Haque, and H. Liu, “*Subspace clustering for high dimensional data: A review*”, *SIGKDD Explorations*, 6(1), 2004, pp.90-105.
15. Lippmann, R.P., “*Evaluating Intrusion Detection Systems: the 1998 DARPA Off-Line Intrusion Detection Evaluation*”, *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, Vol. 2.
16. Ball, G. H. and Hall, D. J., “*ISODATA, a novel method of data analysis and classification*”, *Tech. Rep.. Stanford University*, Stanford, CA, 1965.

Detection and Visualization of Subspace Cluster Hierarchies

Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, Ina Müller-Gorman,
and Arthur Zimek

Institute for Informatics, Ludwig-Maximilians-Universität München, Germany

{achtert, boehm, kriegel, kroegerp, muellerg, zimek}@dbs.ifi.lmu.de

<http://www.dbs.ifi.lmu.de>

Abstract. Subspace clustering (also called projected clustering) addresses the problem that different sets of attributes may be relevant for different clusters in high dimensional feature spaces. In this paper, we propose the algorithm DiSH (Detecting Subspace cluster Hierarchies) that improves in the following points over existing approaches: First, DiSH can detect clusters in subspaces of significantly different dimensionality. Second, DiSH uncovers complex hierarchies of nested subspace clusters, i.e. clusters in lower-dimensional subspaces that are embedded within higher-dimensional subspace clusters. These hierarchies do not only consist of single inclusions, but may also exhibit multiple inclusions and thus, can only be modeled using graphs rather than trees. Third, DiSH is able to detect clusters of different size, shape, and density. Furthermore, we propose to visualize the complex hierarchies by means of an appropriate visualization model, the so-called subspace clustering graph, such that the relationships between the subspace clusters can be explored at a glance. Several comparative experiments show the performance and the effectivity of DiSH.

1 Introduction

The well-known curse of dimensionality usually limits the applicability of traditional clustering algorithms to high-dimensional feature spaces because different sets of features are relevant for different (subspace) clusters. To detect such lower-dimensional subspace clusters, the task of subspace clustering (or projected clustering) has been defined recently. Existing subspace clustering algorithms usually either allow overlapping clusters (points may be clustered differently in varying subspaces) or non-overlapping clusters, i.e. points are assigned uniquely to one cluster or noise. Algorithms that allow overlap usually produce a vast amount of clusters which is hard to interpret. Thus, we focus on algorithms that generate non-overlapping clusters. Those algorithms in general suffer from two common limitations. First, they usually have problems with subspace clusters of significantly different dimensionality. Second, they often fail to discover clusters of different shape and densities, or they assume that the tendencies of the subspace clusters are already detectable in the entire feature space.

A third limitation derives from the fact that subspace clusters may be hierarchically nested, e.g. a subspace cluster of low dimensionality is embedded within several larger subspace clusters of higher dimensionality. None of the existing algorithms is able to

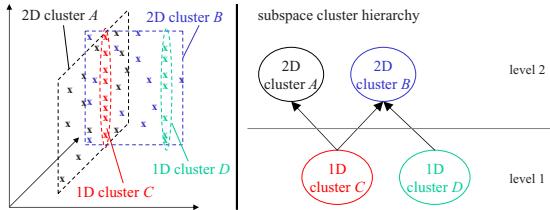


Fig. 1. Hierarchies of subspace clusters with multiple inheritance

detect such important hierarchical relationships among the subspace clusters. An example of such a hierarchy is depicted in Figure 1(left). Two one-dimensional (1D) cluster (C and D) are embedded within one two-dimensional (2D) cluster (B). In addition, cluster C is embedded within both 2D clusters A and B . Detecting such relationships of subspace clusters is obviously a hierarchical problem. The resulting hierarchy is different from the result of a conventional hierarchical clustering algorithm (e.g. a dendrogram). In a dendrogram, each object is placed in a singleton cluster at the leaf level, whereas the root node represents the cluster consisting of the entire database. Any inner node n represents the cluster consisting of the points located in the subtree of n . Dendograms are limited to single inclusion, i.e. a lower dimensional cluster can only be the child cluster of one higher dimensional cluster. However, hierarchies of subspace clusters may exhibit multiple inclusions, e.g. cluster C in Figure 1 is a child of cluster A and B . The concept of multiple inclusions is similar to that of “multiple inheritance” in software engineering. To visualize such more complex relationships among subspace clusters, we need graph representations rather than tree representations. Such a graph representation which we will call *subspace clustering graph* (cf. Figure 1(right)) consists of nodes at different levels. These levels represent the dimensionality of the subspace in which the cluster is found (e.g. the level of cluster A in the graph of Figure 1 is 2). Each object p is assigned to a unique node in that hierarchy representing the lowest dimensional subspace cluster in which p is placed. In addition, an edge between a k -dimensional cluster C and an l -dimensional cluster B , where $l > k$, (e.g. cf. Figure 1) indicates that all points of cluster C are also members of cluster B .

In this paper, we propose the algorithm DiSH (Detecting Subspace cluster Hierarchies) that improves in the following aspects over the state-of-the-art subspace clustering approaches: First, DiSH uncovers complex hierarchies of nested subspace clusters including multiple inclusions. Second, DiSH can detect clusters in subspaces of significantly different dimensionality. Third, DiSH is able to detect clusters of different size, shape, and density. Furthermore, we propose the subspace clustering graph to visualize the resulting complex hierarchies by means of an appropriate visualization model. Using this visualization method the relationships between the subspace clusters can be explored at a glance.

The rest of the paper is organized as follows. We discuss related work in Section 2. Section 3 describes our new algorithm DiSH. The concepts of the clustering graph visualization are outlined in Section 4. An experimental evaluation is presented in Section 5. Section 6 concludes the paper.

2 Related Work

Many subspace clustering algorithms, e.g. [1][2][3][4], aim at finding all clusters in all subspaces of the feature space producing overlapping clusters, i.e. one point may belong to different clusters in different subspaces. In general, these methods also produce some sort of subspace hierarchy. However, those hierarchies are different from the hierarchy addressed in this paper because points are allowed to be placed in clusters such that there are no relationships between the subspaces of these clusters. Thus, the resulting “hierarchy” is much more complex and usually hard to interpret.

Other subspace clustering algorithms, e.g. [5][6][7], focus on finding non-overlapping subspace clusters. These methods assign each point to a unique subspace cluster or noise. Usually, those methods do not produce any information on the hierarchical relationships among the detected subspaces. The only approach to find some special cases of subspace cluster hierarchies introduced so far is HiSC [8]. However, HiSC is limited by the following severe drawbacks. First, HiSC usually assumes that if a point p belongs to a projected cluster C , then C must be visible in the local neighborhood of p in the *entire* feature space. Obviously, this is a quite unrealistic assumption. If p belongs to a projected cluster and the local neighborhood of p in the entire feature space does not exhibit this projection, HiSC will not assign p to its correct cluster. Second, the hierarchy detected by HiSC is limited to single inclusion which can be visualized by a tree (such as a dendrogram). As discussed above, hierarchies of subspace clusters may also exhibit multiple inclusions. To visualize such more complex relationships among subspace clusters, we need graph representations rather than tree representations. Third, HiSC uses a Single-Linkage approach for clustering and, thus, is limited to clusters of particular shapes. DiSH applies a density-based approach similar to OPTICS [9] to the subspace clustering problem that avoids Single-Link effects and is able to find clusters of different size, shape, and densities.

We do not focus on finding clusters of correlated objects that appear as arbitrarily oriented hyperplanes rather than axis-parallel projections (cf. e.g. [10][11][12][13]) because obviously, these approaches are orthogonal to the subspace clustering problem and usually demand more cost-intensive solutions.

3 Hierarchical Subspace Clustering

Let $\mathcal{D} \subseteq \mathbb{R}^d$ be a data set of n feature vectors and \mathcal{A} be the set of attributes of \mathcal{D} . For any subspace $S \subseteq \mathcal{A}$, $\pi_S(o)$ denotes the projection of $o \in \mathcal{D}$ into S . Furthermore, we assume that DIST is a distance function applicable to any $S \subseteq \mathcal{A}$, denoted by DIST^S , e.g. when using the Euclidean distance, $\text{DIST}^S(p, q) = \sqrt{\sum_{a_i \in S} (\pi_{\{a_i\}}(p) - \pi_{\{a_i\}}(q))^2}$.

Our key idea is to define the so-called *subspace distance* that assigns small values if two points are in a common low-dimensional subspace cluster and high values if two points are in a common high-dimensional subspace cluster or are not in a subspace cluster at all. Subspace clusters with small subspace distances are embedded within clusters with higher subspace distances.

For each point $o \in \mathcal{D}$ we first compute the subspace dimensionality representing the dimensionality of that subspace cluster in which o fits best. Thereby, we assume that

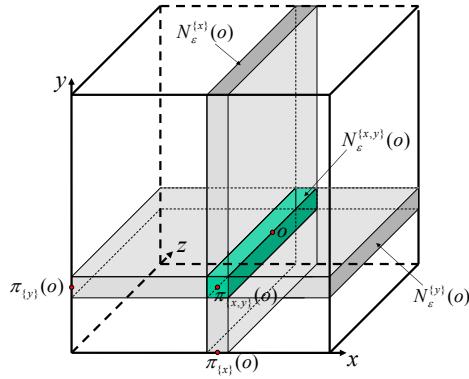


Fig. 2. Subspace selection for a point o (see text for details)

the “best” projection for clustering o is the subspace with the highest dimensionality (providing the most information), or in case of tie-situations, which provides the larger subspace cluster (containing more points in the neighborhood of o w.r.t. the subspace). The subspace dimensionality of a point o is determined by searching for dimensions of low variance (high density) in the neighborhood of o . An attribute-wise ε -range query ($\mathcal{N}_\varepsilon^{\{a_i\}}(o) = \{x \mid \text{DIST}^{\{a_i\}}(o, x) \leq \varepsilon\}$ for each $a_i \in \mathcal{A}$) yields a simple way to assign a predicate to an attribute for a certain object o . If only few points are found within the ε -neighborhood in attribute a_i the variance around o in attribute a_i will be relatively high. For this attribute we will assign 0 as predicate for the query point o , indicating that this attribute does not participate in a subspace that is relevant to any cluster to which o could possibly belong. Otherwise, if $\mathcal{N}_\varepsilon^{\{a_i\}}(o)$ contains at least μ objects, the attribute a_i will be a candidate for a subspace containing a cluster including object o .

From the variance analysis the candidate attributes that might span the best subspace S_o for object o are determined. These attributes need to be combined in a suitable way. This combination problem is equivalent to frequent itemset mining due to the monotonicity $S \subseteq T \Rightarrow |\mathcal{N}_\varepsilon^T(o)| \leq |\mathcal{N}_\varepsilon^S(o)|$. Thus, we can use any frequent itemset mining algorithm (e.g. the Apriori-algorithm [14]) in order to determine the best subspace of an object o .

Definition 1 (subspace preference vector/dimensionality of a point). Let S_o be the best subspace determined for object $o \in \mathcal{D}$. The subspace preference vector $w(o) = (w_1, \dots, w_d)^T$ of o is defined by

$$w_i(o) = \begin{cases} 1 & \text{if } a_i \in S_o \\ 0 & \text{if } a_i \notin S_o \end{cases}$$

The subspace dimensionality $\lambda(o)$ of $o \in \mathcal{D}$ is the number of zero-values in the subspace preference vector $w(o)$.

In the example in Figure 2 the ε -neighborhoods of the 3D point p in attributes x and y are shown by gray-shaded areas. If we assume that both of these areas contain at

least μ points whereas the ε -neighborhood of o along z (not shown) contains less than μ points, o may participate in a subspace cluster that is projected into the subspace $\{x, y\}$. If $|\mathcal{N}_\varepsilon^{\{x,y\}}(o)| \geq \mu$, then $w(o) = (1, 1, 0)^T$ and $\lambda(o) = 1$. Otherwise, none of the 1D subspace clusters containing o can be merged to form a higher dimensional subspace cluster, i.e. we assign o to the subspace containing more points.

Obviously, using any frequent itemset mining algorithm is rather inefficient for high-dimensional data sets, especially when the dimensionality of the subspace clusters are also high-dimensional. Thus, we further propose a heuristics for determining the best subspace S_o for an object o which scales linearly in the number of dimensions. We simply use a best-first search:

1. Determine the candidate attributes of o : $C(o) = \{a_i \mid a_i \in \mathcal{A} \wedge |\mathcal{N}_\varepsilon^{a_i}(o)| \geq \mu\}$.
2. Add $a_i = \arg \max_{a \in C(o)} \{|\mathcal{N}_\varepsilon^a(o)|\}$ to S_o and delete a_i from $C(o)$.
3. Set current intersection $I := \mathcal{N}_\varepsilon^{a_i}(o)$.
4. Determine attribute $a_i = \arg \max_{a \in C(o)} \{|I \cap \mathcal{N}_\varepsilon^a(o)|\}$.
 - (a) If $|I \cap \mathcal{N}_\varepsilon^{a_i}(o)| \geq \mu$ then:
Add a_i to S_o , delete a_i from $C(o)$, and set $I := I \cap \mathcal{N}_\varepsilon^{a_i}(o)$.
 - (b) Else: stop.
5. If $C \neq \emptyset$ continue with Step 4.

Using these heuristics to compute S_o for $o \in \mathcal{D}$, we can determine $w(o)$ as in Definition 1. Overall, we assign a d -dimensional preference vector to each point. The attributes having predicate “1” span the subspace where to find a cluster containing the point, whereas the remaining attributes are irrelevant.

We define a similarity measure between points which assigns a distance of 1, if these two points share a common 1D subspace cluster. If they share a common 2D subspace cluster, they have a distance of 2, etc. This similarity measure is integrated into the algorithm OPTICS [9]. Sharing a common k -dimensional subspace cluster may mean different things: Both points may be associated to the same k -dimensional subspace cluster, or both points may be associated to different $(k-1)$ -dimensional subspace clusters that intersect or are parallel (but not skew). Intuitively, the distance measure between two points corresponds to the dimensionality of the data space which is spanned by the “combined” subspace preference vector of the two points. We first give a definition of the *subspace dimensionality of a pair of points* $\lambda(p, q)$ which follows the intuition of the spanned subspace and then define our subspace distance measure.

Definition 2 (subspace dimensionality of a point pair). *The subspace preference vector $w(p, q)$ of a pair of points $p, q \in \mathcal{D}$ representing the combined subspace of p and q is computed by an attribute-wise logical AND-conjunction of $w(p)$ and $w(q)$, i.e. $w_i(p, q) = w_i(p) \wedge w_i(q)$ ($1 \leq i \leq d$). The subspace dimensionality between two points $p, q \in \mathcal{D}$, denoted by $\lambda(p, q)$, is the number of zero-values in $w(p, q)$.*

We cannot directly use the subspace dimensionality $\lambda(p, q)$ as the subspace distance because points from parallel subspace clusters will have the same subspace preference vector. Thus, we check whether the preference vectors of two points p and q are equal or one preference vector is “included” in the other one. This can be done by computing

the subspace preference vector $w(p, q)$ and checking whether $w(p, q)$ is equal to $w(p)$ or $w(q)$. If so, we determine the distance between the points in the subspace spanned by $w(p, q)$. If this distance exceeds $2 \cdot \varepsilon$, the points belong to different, parallel clusters. The threshold ε , playing already a key role in the definition of the subspace dimensionality (cf. Definition 1), controls the degree of jitter of the subspace clusters.

Since $\lambda(p, q) \in \mathbb{N}$, we usually have many tie situations when merging points/clusters during hierarchical clustering. These tie situations can be solved by considering the distance within a subspace cluster as a second criterion. Inside a subspace cluster the points are then clustered in the corresponding subspace using the traditional OPTICS algorithm and, thus, the subspace clusters can exhibit arbitrary sizes, shapes, and densities.

Definition 3 (subspace distance). Let w be an arbitrary preference vector. Then $S(w)$ is the subspace defined by w and \bar{w} denotes the inverse of w . The subspace distance SDIST between p and q is a pair $SDIST(p, q) = (d_1, d_2)$, where $d_1 = \lambda(p, q) + \Delta(p, q)$ and $d_2 = DIST^{S(\bar{w}(p, q))}(p, q)$, and $\Delta(p, q)$ is defined as

$$\Delta(p, q) = \begin{cases} 1 & \text{if } (w(p, q) = w(p) \vee w(p, q) = w(q)) \wedge DIST^{S(w(p, q))}(p, q) > 2\varepsilon \\ 0 & \text{else,} \end{cases}$$

We define $SDIST(p, q) \leq SDIST(r, s) \iff SDIST(p, q).d_1 < SDIST(r, s).d_1$ or $(SDIST(p, q).d_1 = SDIST(r, s).d_1 \text{ and } SDIST(p, q).d_2 \leq SDIST(r, s).d_2)$.

As suggested in [9], we introduce a smoothing factor μ to avoid the Single-Link effect and to achieve robustness against noise points. The parameter μ represents the minimum number of points in a cluster and is equivalent to the parameter μ used to determine the best subspace for a point. Thus, instead of using the subspace distance $SDIST(p, q)$ to measure the similarity of two points p and q , we use the *subspace reachability* $REACHDIST_\mu(p, q) = \max(SDIST(p, r), SDIST(p, q))$, where r is the μ -nearest neighbor (w.r.t. subspace distance) of p . DiSH uses this subspace reachability and computes a “walk” through the data set, assigning to each point o its smallest subspace reachability with respect to a point visited before o in the walk. The resulting order of the points is called *cluster order*. In a so-called reachability diagram for each point (sorted according to the cluster order along the x -axis) the reachability value is plotted along the y -axis. The valleys in this diagram represent the clusters. The pseudo-code of the DiSH algorithm can be seen in Figure 3.

4 Visualizing Subspace Cluster Hierarchies

The reachability plot is equivalent to tree-like representations and, thus, is not capable of visualizing hierarchies with multiple inclusions. This is illustrated in Figures 4(a) and 4(d). When exploring the reachability plots of the two different data sets A and B, one can see that they look almost the same (cf. Figures 4(b) and 4(e)). Thus, taking only the reachability plots into account, it is impossible to detect the obviously different kind of hierarchy of the second data set. This phenomenon is due to the fact that in data set B we face a subspace cluster hierarchy with multiple inclusion (the 1D cluster is embedded within both 2D clusters).

```

algorithm DiSH( $\mathcal{D}$ ,  $\mu$ ,  $\varepsilon$ )
   $co \leftarrow$  cluster order; // initially empty
   $pq \leftarrow$  empty priority queue ordered by REACHDIST $_{\mu}$ ;
  foreach  $p \in \mathcal{D}$  do
    compute  $w(p)$ ;
     $p.REACHDIST_{\mu} \leftarrow \infty$ ;
    insert  $p$  into  $pq$ ;
  while ( $pq \neq \emptyset$ ) do
     $o \leftarrow pq.\text{next}()$ ;
     $r \leftarrow \mu\text{-nearest neighbor of } o \text{ w.r.t. SDIST}$ ;
    foreach  $p \in pq$  do
      new_sr  $\leftarrow \max(SDIST(o, r), SDIST(o, p))$ ;
       $pq.\text{update}(p, \text{new\_sr})$ ;
    append  $o$  to  $co$ ;
  return  $co$ ;

```

Fig. 3. The DiSH algorithm

This limitation of the reachability plot leads to our contribution of representing the relationships between cluster hierarchies as a so-called subspace clustering graph such that the relationships between the subspace clusters can be explored at a glance. The subspace clustering graph displays a kind of hierarchy which should not be confused with a conventional (tree-like) cluster hierarchy usually represented by dendograms. The subspace clustering graph consists of nodes at several levels, where each level represents a subspace dimension. The top level represents the highest subspace dimension, which has the dimensionality of the data space. It consists of only one root node representing all points that do not share a common subspace with any other point, i.e. the noise points. Let us note that this is different to dendograms where the root node represents the cluster of all objects. The nodes in the remaining levels represent clusters in the subspaces with the corresponding dimensionalities. They are labeled with the preference vector of the cluster they represent. For emphasizing the relationships between the clusters, every cluster is connected with its parents and its children. In contrast to tree representations, like e.g. dendograms, a graph representation allows multiple parents for a cluster. This is necessary, since hierarchical subspace clusters can belong to more than one parent cluster. Consider e.g. data set B, where the objects of the intersection line are embedded in the horizontal plane as well as in the vertical plane, i.e. the cluster forming the intersection line belongs to two parents in the hierarchy. The subspace clustering graphs of the two data sets A and B are depicted in Figures 4(c) and 4(f). The line of data set A is represented by the cluster with the preference vector [1,0,1]. This cluster is a child of cluster [1,0,0] representing the plane in data set A (cf. Figure 4(c)). The more complex hierarchy of data set B is represented in Figure 4(f), where the cluster [1,0,1] belongs to two parent clusters, the cluster of the horizontal plane [0,0,1] and the cluster of the vertical plane [1,0,0].

In contrast to dendograms, objects are not placed in singleton clusters at the leaf level, but are assigned to the lowest-dimensional subspace cluster they fit in within

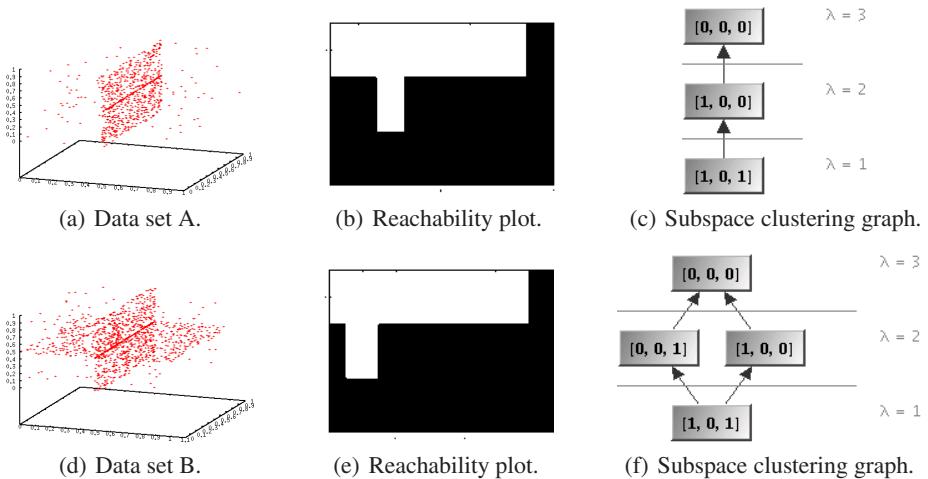


Fig. 4. Different hierarchies in 3-dimensional data

```

method extractCluster(ClusterOrder co)
  cl ← empty list; // cluster list
  foreach o ∈ co do
    p ← o.predecessor;
    if (∀c ∈ cl with w(c) = w(o,p) ∧ distw(o,p)(o, c.center) ≤ 2 · ε) then
      create a new c;
      add c to cl;
      add o to c;
  return cl;

```

Fig. 5. The method to extract the clusters from the cluster order

the graph. Similar to dendrograms, an inner node n of the subspace clustering graph represents the cluster of all points that are assigned to n and of all points assigned to its child nodes.

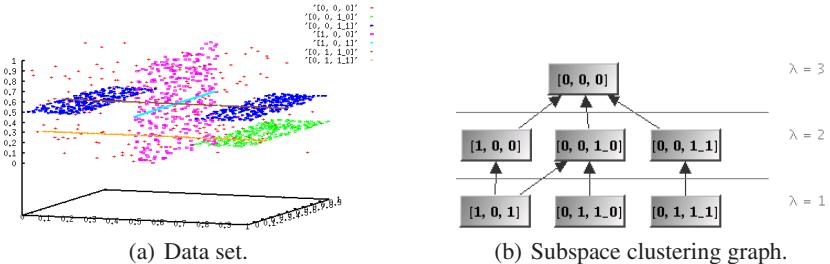
To build the subspace clustering graph, we extract in a first step all clusters from the cluster order. For each object o in the cluster order the appropriate cluster c has to be found, where the preference vector $w(c)$ of cluster c is equal to the preference vector $w(o,p)$ between o and its predecessor p . Additionally, since parallel clusters share the same preference vector, the weighted distance between the centroid of the cluster c and object o with $w(o,p)$ as weighting vector has to be less or equal to 2ϵ . The complete method to extract the clusters from the cluster order can be seen in Figure 5.

After the clusters have been derived from the cluster order, the second step builds the subspace cluster hierarchy. For each cluster we have to check, if it is part of one or more (parallel) higher-dimensional clusters, whereas each cluster is at least the child of the noise cluster. The method to build the subspace hierarchy from the clusters is depicted in Figure 6.

```

method buildHierarchy ( $c_l$ )
   $d \leftarrow$  dimensionality of objects in  $\mathcal{D}$ ;
  foreach  $c_i \in c_l$  do
    foreach  $c_j \in c_l$  do
      if ( $\lambda_{c_j} > \lambda_{c_i}$ ) then
         $d \leftarrow dist_w(c_i, c_j)(c_i.\text{center}, c_j.\text{center})$ ;
        if ( $\lambda_{c_j} = d \vee (d \leq 2 \cdot \varepsilon \wedge \#\{c \in c_i.\text{parents} : \lambda_c < \lambda_{c_j}\})$ ) then
          add  $c_i$  as child to  $c_j$ ;

```

Fig. 6. The method to build the hierarchy of subspace clusters**Fig. 7.** Results on synthetic dataset DS1**Table 1.** Runtime, precision and recall w.r.t. the strategy for preference vector computation

	APRIORI			BEST-FIRST		
	DS1	DS2	DS3	DS1	DS2	DS3
runtime [sec]	147	32	531	76	14	93
precision [%]	99.7	99.5	99.7	99.7	99.5	99.5
recall [%]	99.8	99.6	99.8	99.8	99.6	99.5

5 Experimental Evaluation

We first evaluated DiSH on several synthetic data sets. Exemplary, we show the results on three data sets named “DS1”, “DS2”, and “DS3”.

We evaluated the precision, recall and the runtime of our DiSH algorithm w.r.t. the strategies used for determination of the preference vectors. The strategy using the Apriori-algorithm [14] is denoted with “APRIORI”, the heuristics using the best-first search is denoted with “BEST-FIRST”. The results of the runs with both strategies on the three data sets are summarized in Table II. Since the heuristics using best-first search outperforms the strategy using the Apriori-algorithm in terms of runtime and has almost equal precision and recall values, we used in all further experiments the heuristics to compute the preference vectors rather than the Apriori-based approach.

Data set “DS1” (cf. Figure 7(a)) contains 3D points grouped in a complex hierarchy of 1D and 2D subspace clusters with several multiple inclusions and additional noise

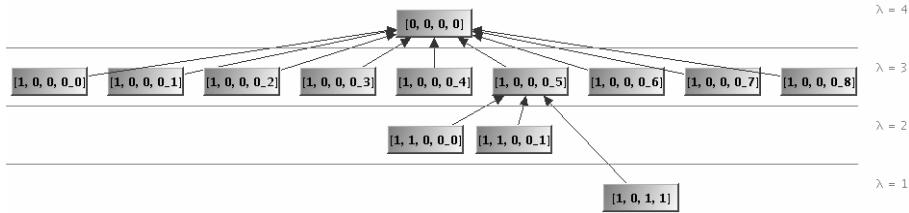


Fig. 8. Subspace clustering graph of the Forest data

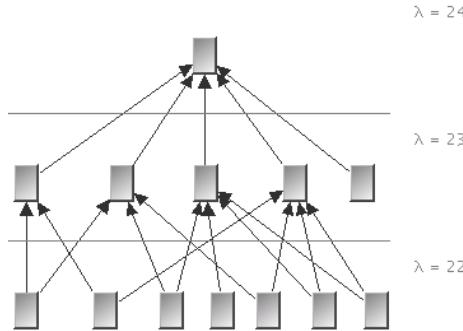


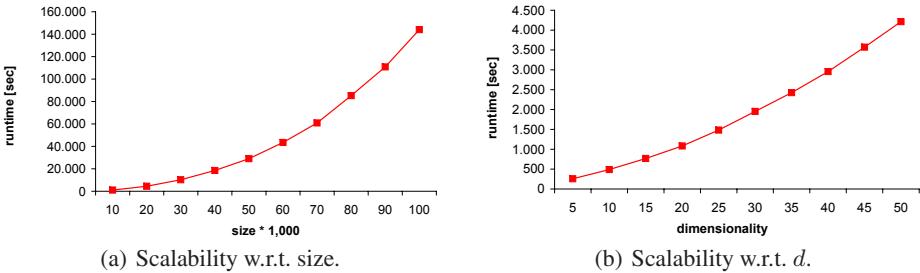
Fig. 9. Subspace clustering graph of the Gene data

points. The results of DiSH applied to DS1 are depicted in Figure 7(b). As it can be seen, the complete hierarchical clustering structure can be obtained from the resulting subspace clustering graph. In particular, the complex nested clustering structure can be seen at a glance. Data set “DS2” is a 5D data set containing ten clusters of different dimensionality and noise: one cluster is embedded in a 4D subspace, four clusters are 3D, three clusters are 2D and two clusters are 1D subspace clusters. The resulting subspace clustering graph (not shown due to space limitations) produced by DiSH exhibits all ten subspace clusters of considerably different dimensionality correctly. Similar observations can be made when evaluating the subspace clustering graph obtained by DiSH on data set “DS3” (not shown due to space limitations). The 16D data set DS3 contains noise points, one 13 dimensional, one 11 dimensional, one 9 dimensional, one 7 dimensional cluster, and two 6 dimensional clusters. Again, DiSH found all six subspace clusters correctly.

We also applied HiSC, PreDeCon and PROCLUS on DS1 for comparison. Neither PreDeCon nor PROCLUS are able to detect the hierarchies in DS1 and the subspace clusters of significantly different dimensionality. HiSC performed better in detecting simple hierarchies of single inclusion but fails to detect multiple inclusions.

In addition, we evaluate DiSH using several real-world data sets. Applied to the Wisconsin Breast Cancer Database (original) from the UCI ML Archive¹ ($d = 9$, $n = 569$, objects labeled as “malignant” or “benign”) DiSH finds a hierarchy containing

¹ <http://www.ics.uci.edu/~mlearn/MLSummary.html>

**Fig. 10.** Scalability results

several low dimensional clusters and one 7D cluster ($\varepsilon = 0.01, \mu = 15$). An additional 9D cluster contains the noise points. It is worth mentioning that the reported clusters are pure. In particular, the seven low dimensional clusters only contain objects labeled as “benign”, whereas the 7D cluster only contains objects marked as “malignant”.

We applied DiSH on the Wages data set² ($d = 4, n = 534$). Since most of the original attributes are not numeric, we used only 4 dimensions (YE=years of education, W=wage, A=age, and YW=years of work experience) for clustering. The resulting subspace cluster hierarchy (using $\varepsilon = 0.001, \mu = 9$) is visualized in Figure 8. The nine parallel clusters having a subspace dimensionality of $\lambda = 3$ consist of data of people having equal years of education, e.g. in cluster $[1, 0, 0, 0..0]$ YE=17 and in cluster $[1, 0, 0, 0..5]$ YE=12. The two clusters labeled with $[1, 1, 0, 0..0]$ and $[1, 1, 0, 0..1]$ in the 2D subspace are children of cluster $[1, 0, 0, 0..5]$ and have (in addition to equal years of education, YE=12) equal wages values (W=7.5 and W=5, respectively). The 1-dimensional cluster $[1, 0, 1, 1]$ is a child of $[1, 1, 0, 0..0]$ and has the following properties: YE=12, A=26, and YW=8.

Last but not least, we applied DiSH to the yeast gene expression data set of [15] ($d = 24, n \approx 4,000$). The result of DiSH (using $\varepsilon = 0.01, \mu = 100$) on the gene expression data is shown in Figure 9. Again, DiSH found several subspace clusters of different subspace dimensionalities with multiple inclusions.

The scalability of DiSH w.r.t. the data set size is depicted in Figure 10(a). The experiment was run on a set of 5D synthetic data sets with increasing number of objects ranging from 10,000 to 100,000. The objects are distributed over equally sized subspace clusters of subspace dimensionality $\lambda = 1, \dots, 4$ and noise. As parameters for DiSH we used $\varepsilon = 0.001$ and $\mu = 20$. As it can be seen, DiSH scales slightly super-linear w.r.t. the number of tuples. A similar observation can be made when evaluating the scalability of DiSH w.r.t. the dimensionality of the data set (cf. Figure 10(b)). The experiments were obtained using data sets with 5,000 data points and varying dimensionality of $d = 5, 10, 15, \dots, 50$. For each data set the objects were distributed over $d - 1$ subspace clusters of subspace dimensionality $\lambda = 1, \dots, d - 1$ and noise. Again, the result shows a slightly superlinear increase of runtime when increasing the dimensionality of the data set. The parameters for DiSH were the same as in the evaluation of the scalability of DiSH w.r.t. the data set size ($\varepsilon = 0.001$ and $\mu = 20$).

² http://lib.stat.cmu.edu/datasets/CPS_85_Wages

6 Conclusions

In this paper, we presented DiSH, the first subspace clustering algorithm for detecting complex hierarchies of subspace clusters. DiSH is superior to the state-of-the-art subspace clustering algorithms in several aspects: First, it can detect clusters in subspaces of significantly different dimensionality. Second, it is able to determine hierarchies of nested subspace clusters containing single and multiple inclusions. Third, it is able to detect clusters of different size, shape, and density. Fourth, it does not assume that the subspace preference of a point p is exhibited in the local neighborhood of p in the entire data space. We have shown by performing several comparative experiments using synthetic and real data sets that DiSH has a superior performance and effectiveness compared to existing methods.

References

1. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: Proc. SIGMOD. (1998)
2. Cheng, C.H., Fu, A.W.C., Zhang, Y.: Entropy-based subspace clustering for mining numerical data. In: Proc. KDD. (1999) 84–93
3. Kailing, K., Kriegel, H.P., Kröger, P.: Density-connected subspace clustering for high-dimensional data. In: Proc. SDM. (2004)
4. Kriegel, H.P., Kröger, P., Renz, M., Wurst, S.: A generic framework for efficient subspace clustering of high-dimensional data. In: Proc. ICDM. (2005)
5. Aggarwal, C.C., Procopiuc, C.M., Wolf, J.L., Yu, P.S., Park, J.S.: Fast algorithms for projected clustering. In: Proc. SIGMOD. (1999)
6. Procopiuc, C.M., Jones, M., Agarwal, P.K., Murali, T.M.: A Monte Carlo algorithm for fast projective clustering. In: Proc. SIGMOD. (2002)
7. Böhm, C., Kailing, K., Kriegel, H.P., Kröger, P.: Density connected clustering with local subspace preferences. In: Proc. ICDM. (2004)
8. Achtert, E., Böhm, C., Kriegel, H.P., Kröger, P., Müller-Gorman, I., Zimek, A.: Finding hierarchies of subspace clusters. In: Proc. PKDD. (2006) To appear.
9. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: OPTICS: Ordering points to identify the clustering structure. In: Proc. SIGMOD. (1999)
10. Yang, J., Wang, W., Wang, H., Yu, P.S.: Delta-Clusters: Capturing subspace correlation in a large data set. In: Proc. ICDE. (2002)
11. Wang, H., Wang, W., Yang, J., Yu, P.S.: Clustering by pattern similarity in large data sets. In: Proc. SIGMOD. (2002)
12. Böhm, C., Kailing, K., Kröger, P., Zimek, A.: Computing clusters of correlation connected objects. In: Proc. SIGMOD. (2004)
13. Aggarwal, C.C., Yu, P.S.: Finding generalized projected clusters in high dimensional space. In: Proc. SIGMOD. (2000)
14. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. SIGMOD. (1994)
15. Spellman, P.T., Sherlock, G., Zhang, M.Q., Iyer, V.R., Anders, K., Eisen, M.B., Brown, P.O., Botstein, D., Futcher, B.: "Comprehensive Identification of Cell Cycle-Regulated Genes of the Yeast *Saccharomyces cerevisiae* by Microarray Hybridization.". Molecular Biology of the Cell **9** (1998) 3273–3297

Correlation-Based Detection of Attribute Outliers

Judice L.Y. Koh^{1,2}, Mong Li Lee², Wynne Hsu², and Kai Tak Lam³

¹ Institute for Infocomm Research, Singapore 119613

² School of Computing, National University of Singapore

³ Institute of High Performance Computing, Singapore 117528

judice@i2r.a-star.edu.sg, {leeml,whsu}@comp.nus.edu.sg,
lamkt@ihpc.a-star.edu.sg

Abstract. An outlier is an object that does not conform to the normal behavior of the data set. In data cleaning, outliers are identified for data noise reduction. In applications such as fraud detection, and stock market analysis, outliers suggest abnormal behavior requiring further investigation. Existing outlier detection methods have focused on class outliers and research on attribute outliers is limited, despite the equal role attribute outliers play in depreciating data quality and reducing data mining accuracy. In this paper, we propose a novel method to detect attribute outliers from the deviating correlation behavior of attributes. We formulate three metrics to evaluate outlier-ness of attributes, and introduce an adaptive factor to distinguish outliers from non-outliers. Experiments with both synthetic and real-world data sets indicate that the proposed method is effective in detecting attribute outliers.

Keywords: Outlier detection, Data cleaning.

1 Introduction

An outlier is an object exhibiting alternative behavior in a data set. It is a data point that does not conform to the general patterns characterizing the data set. Detecting outliers has important applications in data cleaning as well as in the mining of abnormal patterns for fraud detection, stock market analysis, intrusion detection, marketing, network sensors, email spam detection, among others.

There are two types of outliers, the class and the attribute outliers [1]. A class outlier is a multivariate data point (tuple) which does not fit into any class by definitions of distance, density, or nearest-neighbor. An attribute outlier, in general sense, is an external error introduced to the attribute values. In this paper, we formally define attribute outlier as a *univariate point which exhibits deviating correlation behavior with respect to other attributes*.

Existing outlier detection methods focus primarily on class outliers, although for a number of reasons, detecting attribute outliers is an equally important data mining problem. First, class outliers are often the result of one or more attribute outliers. Correcting or eliminating only the affecting attributes rather than the tuples has the advantage of fixing the abnormal behaviors while retaining information. Second, even when attribute outliers do not affect class memberships, they may still interfere with the data analysis mechanisms as data noise. Third, for many real-world data sets that do not contain class attributes, it is still meaningful to identify attribute outliers which

are sources of errors. One example is the UniProt database which contains the functional, structural, and physico-chemical descriptions of proteins [2]. Though there is no meaningful class attribute for proteins, maintaining correctness of every detail provided in these records is critical, given that they are extensively referenced by the world-wide biological researchers for analysis and experimental planning.

Since attribute outliers do not arise from the context of class outliers, they cannot be defined from the view point of the latter. The nature of problems associated with class and attribute outliers differ and separate detection methods are needed. We propose a novel correlation-based approach for attribute outlier detection in data subspaces. We call the outlier detection method **ODDS** to denote attribute **O**utlier **D**etection from **D**ata **S**ubspaces. Specific contributions of this paper include:

1. A formal definition of attribute outliers based on the correlation behavior of attributes in data subspaces.
2. Three new metrics *O-measure*, *P-measure* and *Q-measure* to quantify the deviating correlation behavior of an attribute. O-measure is the most accurate while Q-measure is computationally less intensive. P-measure is devised for sparse data sets containing vast occurrences of rare attribute values which are not outliers.
3. An adaptive *Rate-of-change* factor for the selection of optimal thresholds that distinguishes the outliers from non-outliers in any given data set. These automatic and data-dictated thresholds remove dependency on user-defined parameter.
4. The ODDS algorithm which systematically detects attribute outliers in data subspaces, and two filtering strategies to quickly identify subspaces that do not contain attribute outliers.

The rest of this paper is organized as follows. A motivating example is given in the next section. Related works are discussed in Section 3. Formal definitions are detailed in Section 4. In Section 5, we present the ODDS algorithm. Experimental evaluations are presented in Section 6, and we conclude in Section 7.

2 Motivating Example

We first illustrate the rationale of our deviation metrics for attribute outlier detection using the example in Table 1 and Figure 1.

Table 1. World Clock data set containing 4 attribute outliers. W, Y and Z are erroneous entries, while X is an uncommon abbreviation of ‘British Columbia’.

	Country	State	City	Day	Time [†]	Weather
1	U.S.A	California	LA	Tue	8:40pm	Sunny
2	U.S.A	California	LA	Tue	8:40pm	Rainy
3	U.S.A	California	Vancouver ^Y	Wed ^Z	8:40pm	Sunny
4	U.S.A	California	LA	Tue	8:40pm	Storm
5	U.S.A	California	LA	Tue	8:40pm	Snow
6	Canada	British Columbia	Vancouver	Tue	8:40pm	Storm
7	Canada	British Columbia	Vancouver	Tue	8:40pm	Sunny
8	Canada	California ^W	Vancouver	Tue	8:40pm	Rainy
9	Canada	B.C. ^X	Vancouver	Tue	8:40pm	Rainy
10	Canada	British Columbia	Vancouver	Tue	8:40pm	Rainy
11	Micronesia	Ponape	Palikir	Wed	2:40pm	Storm

[†] Class attribute ^{W, X, Y, Z} Attribute outliers

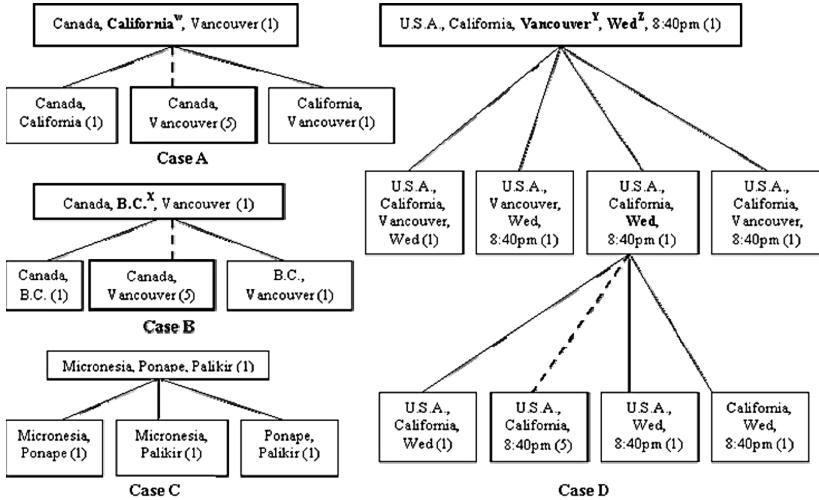


Fig. 1. Selected attribute combinations of the World Clock dataset and their supports

First, we observed that tuples with one or more *rare values* may possibly be class outliers, but *for attribute outliers, rarity does not equate abnormality*. Consider Case C in Figure 1 – the tuple is a perfectly legitimate class outlier belonging to the *rare class* of ‘2:40pm’ in Table 1. However, the attributes of ‘Micronesia’, ‘Ponape’ and ‘Palikir’, though rare in individual dimensions of Country, State and City, are consistent in their correlation behavior and are not erroneous. In a similar example, 3 out of 208,005 tuples in the UniProt protein database (Release 7.1) contain the values <‘Parkin’, ‘PKRN’, ‘S-nitrosylation’> for attributes *Protein name*, *Gene name* and *Keyword* respectively. Despite rarity in their dimensions, they are not attribute outliers. In reality, few known protein sequences are associated with the Parkinson disease, but they are consistently known as Parkin, are products of PKRN gene, and are post-translationally modified by nitrosylation.

Rarity may be a good indicator for class outlier-ness. But for attribute outliers, observations should be drawn from the correlation behavior of attributes. Consider Case A – while ‘Vancouver’ and ‘Canada’ co-occur in five tuples, only one sub-tuple of <‘Canada’, ‘California’> and of <‘California’, ‘Vancouver’> exist. Intuitively, greater differences in the sub-tuple supports indicate higher likelihood that ‘California’ is an outlier in combination <‘Canada’, ‘California’, ‘Vancouver’>. This forms the basis of our outlier metrics. The same analogy identifies X in Case B.

In certain sparse data sets such as the UniProt database, finding the vast occurrences of rare attribute values such as ‘B.C’ in Case B is not of prime interest. Unlike ‘California’ in Case A, ‘B.C’ is not necessarily erroneous. Therefore, the *P-measure* metric is designed to disfavor rare values from attribute outlier detection.

In real-world databases, a tuple often contains multiple attribute outliers. Due to the interferences of the correlation patterns, it is difficult to determine multiple attribute outliers from an attribute combination. However, an attribute outlier can be isolated at lower dimensional attribute combinations. Consider Case D – the two attribute outliers are separated when they are projected into different 4-attribute sub-tuples.

Our proposed ODDS algorithm systematically iterates through data subspaces which are projected relations of two or more attributes. Distinguishing attribute outliers as *local deviators in data subspaces* also has the benefit of eliminating interferences of noisy and uncorrelated dimensions with the outlier detection algorithms. For example, the Weather dimension in Table 1 does not relate to any other attributes but contain non-deterministic/random values interfering with the outlier detection mechanisms.

To isolate the attribute outliers from non-outliers, users typically need to define a threshold. This is not viable in practice, given that the number of outliers in the real world dataset varies depending on the noise level of the data set and the data dimension under study. In the ODDS algorithm, the optimal threshold is determined from the maximal *Rate-of-change* which intuitively marks the point where sorted outlier scores drastically change. Rate-of-change is the natural boundary separating the outliers and non-outliers, and it removes the dependency of the outlier detection on any user-specified parameter.

3 Related Works

Among the few attribute outlier detection methods are distribution-based approaches that eliminates attribute values that do not fit into the distribution models of the data set [3, 4]. Accuracy of distribution-based methods largely depends on the best-fit distribution models used, and they are limited to finding obvious off-scale values.

Data polishing approaches to attribute outlier detection problem construct for each dimension a classifier based on the remaining dimensions and the class dimension [1, 5]. Incorrect predictions are flagged as attribute outliers. The accuracy varies depending on the classifier used and they mainly focused on attribute outliers resulting in change of class membership.

Class outlier detection methods have been extensively studied. Clustering-based algorithms generate outliers as “miniature” clusters, either though optimizing cluster size and relative distance from neighboring clusters [6], or eliminating clusters at longest edges of a Minimum Spanning Tree (MST) [7]. These methods generally suffer from expediting cost as data dimensionality and size increases.

Density-based class outlier detection methods measure the number of tuples in the surrounding neighborhoods [8]. Because of the large number of k-nearest neighbor queries, computational cost is high but may be reduced through pruning mechanisms [9, 10]. They are restricted to continuous data sets measurable by proximities.

Distance-based approaches define a class outlier by the β fraction of other data points which are less than κ distance from it [11]. Native methods do not scale well with data dimensionality and size but this can be reduced by pruning in data partitions or p-tree data structures [12, 13]. Also, the accuracy of distance-based methods is highly dependent on the user parameters β and κ . Too high β leads to more false positives while low κ causes more false negatives.

Comparatively, the proposed ODDS method is applicable to categorical data, and can be extended to continuous data by discretizing the values into bins. Further, the ODDS method is parameter-less; the thresholds are determined using an adaptive factor generated from the data set.

4 Definitions

In this section, we formalize the notion of an attribute outlier and give definitions of the metrics used in our algorithm.

Definition 1 (Support). Let R be a relation with m attributes A_1, A_2, \dots, A_m . Let S be a projection of degree $(v-u+1)$ on R over attributes A_u, \dots, A_v , $S = \pi_{A_u, \dots, A_v}(R)$. The support of a tuple s in S , denoted by $\text{sup}(s)$, is the count of the tuples in R that have the same values for attributes A_u, \dots, A_v as tuple s .

For example, consider the World-Clock relation $R(\text{Country}, \text{State}, \text{City}, \text{Day}, \text{Time}, \text{Weather})$ in Table 1, and a projected relation over three attributes, $S = \pi_{\text{Country}, \text{State}, \text{City}}(R)$. The support of tuple $\langle \text{'U.S.A.'}, \text{'California'}, \text{'LA'} \rangle$ in S is 4 since tuples 1, 2, 4 and 5 in R have the same attribute values for Country, State and City. Similarly, $\text{sup}(\langle \text{'Canada'}, \text{'California'}, \text{'Vancouver'} \rangle) = 1$.

Definition 2 (Neighborhood). Let tuple $s = \langle a_u, \dots, a_v \rangle$. Without loss of generality, we consider A_v as the target attribute whose extent of deviation we are interested to determine. The neighborhood of A_v w.r.t s is defined as $N(A_v, s) = \langle a_u, \dots, a_{v-1} \rangle$. The support of $N(A_v, s)$ is the count of tuples in R with the same values a_u, \dots, a_{v-1} for A_u, \dots, A_{v-1} .

Continuing from the last example, consider tuple $s = \langle \text{'Canada'}, \text{'California'}, \text{'Vancouver'} \rangle$ in the projected relation S . The neighborhood of the State attribute in tuple s , denoted as $N(\text{State}, s)$ is the sub-tuple $\langle \text{'Canada'}, \text{'Vancouver'} \rangle$. Since the same values of ‘Canada’ and ‘Vancouver’ for attributes Country and City respectively are found in tuples 6, 7, 8, 9 and 10 of R , we have $\text{sup}(N(\text{State}, s)) = 5$.

Our objective is to determine attributes which deviate from its neighbors in the projected relations. We formulate three metrics O-measure, P-measure and Q-measure to quantify the extent of deviation.

Definition 3 (O-measure). The O-measure (Outlier measure) of target attribute A_v w.r.t. s is defined as

$$O\text{-measure}(A_v, s) = \frac{\sum_{i=u}^{v-1} \text{sup}(N(A_i, s))}{\text{sup}(N(A_v, s))} \quad (1)$$

The lower the O-measure score, the more likely attribute A_v is an attribute outlier in s . Let us compute the O-measure of the attribute outlier W in Table 1. Let $s = \langle \text{'Canada'}, \text{'California'}, \text{'Vancouver'} \rangle$ be a tuple of $S = \pi_{\text{Country}, \text{State}, \text{City}}(R)$. The support of $N(\text{State}, s)$ is 5 while $\text{sup}(N(\text{Country}, s))$ and $\text{sup}(N(\text{City}, s))$ are both 1. The O-measure of the State attribute w.r.t. s is $(1+1)/5=0.4$.

For comparison, we also compute the O-measure of the State attribute in tuple $t = \langle \text{'U.S.A.'}, \text{'California'}, \text{'LA'} \rangle$. We have $O\text{-measure}(\text{State}, t) = (\text{sup}(N(\text{Country}, t)) + \text{sup}(N(\text{City}, t))) / \text{sup}(N(\text{State}, t)) = (4+5)/4 = 2.25$. ‘California’ is an attribute outlier in attribute combination s but not in t , therefore $O\text{-measure}(\text{State}, s)$ is relatively lower than $O\text{-measure}(\text{State}, t)$. Recall that the outlier metric should not consider rare classes or events as attribute outliers. This is evident using O-measure where the high $O\text{-measure}(\text{Country}, \langle \text{'Micronesia'}, \text{'Ponape'}, \text{'Palikir'} \rangle) = 2$ prevents the mis-interpretation of Micronesia as an attribute outlier.

Definition 4 (P-measure). Let $\text{freq}(A_v)$ be the frequency of an attribute A_v in the original relation R. The P-measure of A_v w.r.t a tuple s is defined as

$$P\text{-measure}(A_v, s) = \frac{\sum_{i=u}^{v-1} \sup(N(A_i, s))}{\sup(N(A_v, s)) \text{freq}(A_v)} \quad (2)$$

P-measure takes into account the support of A_v in R. A lower weightage is given to the rare attribute values which have lower frequencies. Unlike O-measure, P-measure favors non-rare values and is more effective in identifying attribute outliers in sparse data set which contained vast occurrences of rare attribute values which are not attribute outliers.

Consider the attribute outlier X in Table 1. Given the low frequency of the value ‘B.C.’ in the data set, the low O-measure score almost guarantee that the State attribute in s= <‘Canada’, ‘B.C.’, ‘Vancouver’> will be labeled as an outlier, that is, $O\text{-measure}(\text{State}, s) = (1+1)/4 = 0.5$. In contrast, $P\text{-measure}(\text{State}, s) = (1+1)/(4*0.09) = 5.6$ is relatively much higher.

Definition 5 (Q-measure). The Q-measure of an attribute A w.r.t tuple s is defined as

$$Q\text{-measure}(A, s) = \frac{\sup(s)}{\sup(N(A, s))} \quad (3)$$

Let a be the attribute value of A. Q-measure is the conditional probability of a tuple having the value a for attribute A, given that the tuple has the same attribute values as the neighborhood of A. Relating this back to the attribute outlier W in Table 1, $Q\text{-measure}(\text{State}, <\text{‘Canada’}, \text{‘California’}, \text{‘Vancouver’}>) = 1/5 = 0.2$.

Computationally, it is less intensive to use Q-measure as the outlier detection metric because less calculation of the supports of neighborhoods is required. This is however, at the expense of accuracy performance which we will show in Section 6.

Definition 6 (CA-Outlier). Let S be relation of n tuples $S=\{s_1, \dots, s_n\}$. Given a threshold β , a Correlation-based Attribute (CA-)outlier is a paired set (A, s_i) , $1 \leq i \leq n$ such that the deviation scores of A w.r.t s_i based on an outlier metric is less than β . Optimal value of β can be automatically derived using Rate-of-change.

Definition 7 (Rate-of-change). Given an attribute A and the set of $O\text{-measure}(A, s_i)$ $\forall s_i \in S$, $1 \leq i \leq n$. Let L be the list of tuples s_i sorted in ascending order of $O\text{-measure}(A, s_i)$. The Rate-of-change of a ranked tuple s_i ($2 \leq i \leq n$) is defined as

$$\text{Rate - of - change}(s_i) = \frac{O\text{-measure}(A, s_i) - O\text{-measure}(A, s_{i-1})}{O\text{-measure}(A, s_{i-1})} \quad (4)$$

The same formula can be applied to determine the Rate-of-change based on the P-measure and Q-measure metrics.

5 Algorithms

We regard attribute outlier as a local deviator which exhibits alternative correlation behavior in a data subspace. Consider a relation R with m attributes and n tuples. In the worst case, scanning all data subspaces (or projected relations) in R require $O(n \times 2^m)$ searches where 2^m is the total number of projected relations on R. Therefore,

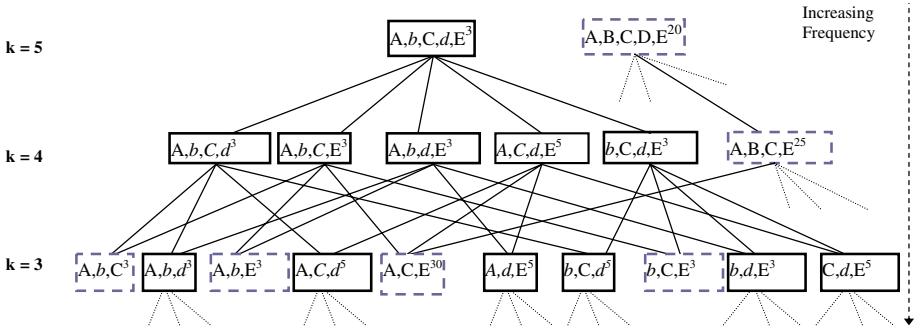


Fig. 2. Attribute combinations at projections of degree k with attribute outliers b and d . The numerical values at the top right corner of the combinations are the corresponding supports.

computing the O-measure scores for each attribute w.r.t every projected relations requires $O(2^m \times n \times m)$ time complexity. Obviously, the brute-force approach of searching every data subspaces of a relation for CA-outliers is highly inefficient. To overcome this limitation, we propose two filtering strategies to identify and prune data subspaces that cannot possibly contain an attribute outlier.

Figure 2 shows the attribute combinations in a relation of 5 attributes. We assume that all possible projections of the relation are completely enumerated. Intuitively, a frequent tuple of any projected relation cannot be a CA-outlier. Our first strategy filters any tuple s with $\text{sup}(s) \geq \text{minsup}$, s and its sub-tuples from the calculation of the outlier scores. Pruning of sub-tuples follows the *Apriori property*: supports of sub-tuples increase across projected relations of decreasing degrees. For example, $\text{sup}(< 'A', 'b', 'C', 'd', 'E' >) \leq \text{sup}(< 'A', 'C', 'd', 'E' >) \leq \text{sup}(< 'A', 'C', 'E' >)$. In Figure 2, setting minsup at 20 will prune off $< 'A', 'B', 'C', 'D', 'E' >$ with sub-tuples $< 'A', 'B', 'C', 'E' >$ and $< 'A', 'C', 'E' >$.

The second filtering strategy only applies to the Q-measure metric which exhibits the monotone property. We prove that if ' b ' is a CA-outlier in a tuple s based on Q-measure, it is also CA-outlier in the sub-tuples of s .

Property 1. Let s be a tuple in projected relation S . An attribute A is a CA-outlier w.r.t s based on Q-measure implies that A is a CA-outlier w.r.t any sub-tuple of s which also contains A .

Proof. Let b be a CA-outlier w.r.t $s = < 'A', 'b', 'C', 'D' >$ detected using the Q-measure deviation metric. Let s' be a sub-tuple of s . Let β be the optimal threshold such that for any CA-outlier A , $Q\text{-measure}(A, s) \leq \beta$. Based on the *Apriori property*, we have

$$\text{sup}(N(b, s)) \leq \text{sup}(N(b, s'))$$

$$Q\text{-measure}(b, s') = \frac{\text{sup}(b)}{\text{sup}(N(b, s'))} \leq \frac{\text{sup}(b)}{\text{sup}(N(b, s))} = Q\text{-measure}(b, s) \leq \beta$$

Hence, b is also a CA-outlier in s' . Sub-tuples of any CA-outlier found using Q-measure in an attribute combination are eliminated from deviation computation. In Figure 2, sub-tuples $< 'A', 'b', 'C' >$, $< 'A', 'b', 'E' >$ and $< 'b', 'C', 'E' >$ are omitted when

'b' is detected a CA-outlier in <'A','b','C','E'>. Beyond reducing the time complexity of the outlier score calculation; this property enables reduction of the time for enumerating the projections.

Algorithm 1 shows the details of the ODDS algorithm. A top-down iteration over the data subspaces, starting from the original relation R to the projected relations at level 3 is performed [line 6]. The tuples or attribute combinations are stored into a list L. Iteration begins by eliminating tuples which have frequency or supports greater than *minsup* from L [line 4-6]. The program terminates if the list L is empty [line 8].

The metrics are computed for each target attribute of each attribute combinations at level i [line 10-20]. Note that for Q-measure, it is not necessary to iterate over neighborhoods (unlike O-measure or P-measure) of the target attribute and is therefore computational cheaper. Further, Property 1 provides effective pruning for Q-measure. Tuples belonging to the (i-1) data subspaces are generated from the existing tuples in L at the end of each iteration [line 25-26]. For each dimension, Get_CA-outliers function accepts a list of all attributes of the same dimension and its corresponding O-, P-, or Q-measure values. These attribute points are sorted in ascending values of their deviation scores [line 1, Get_CA-outliers] to identify the maximum Rate-of-change [line 5, Get_CA-outliers]. Attribute points above max Rate-of-change are output as CA-outliers [line 6-8].

```

ALGORITHM 1. ODDS Attribute Outliers Detection method
INPUT: Enumerated projections of degree 1..k for relation R with m
       attributes. User input minsup.
OUTPUT: CA-outliers and the corresponding tuples of projected relations
1. Initialize List L
2. Insert into L projected relations of degree m of R
3. From degree i = m to 3 do
4.   For each tuple s in L
5.     Remove s if sup(s) ≥ minsup
6.   Endfor
7.   If L is empty then
8.     Terminate program //end data subspace.
9.   EndIf
10.  For each tuple s in L do
11.    For each target attribute A in s do
12.      Q-measure(A,s)=sup(s)/sup(N(A,s))
13.      O-measure(A,s)=0
14.      For each neighbors Ci of A do
15.        O-measure(A,s)=O-measure(A,s)+sup(N(Ci,s))
16.      Endfor
17.      O-measure(A,s)=O-measure(A)/sup(N(A,s))
18.      P-measure(A,s)=O-measure(A,s)/sup(A)
19.    Endfor
20.  Endfor
21.  For each target A do // compute Rate-of-change
22.    OUTPUT Get_CA-outliers (si)
23.  Enddo
24.  Remove in L sub-tuples of the detected CA-outliers // Q-measure
25.  Extract all tuples s in L
26.  Insert into L sub-tuples of s of degree i-1
27. Endfor
FUNCTION. Get_CA-outliers
INPUT: List of attributes Aj and subsets with O-, P- or Q-measure values
OUTPUT: CA-outliers according to adaptive Rate-of-change thresholds
1. B ← Aj sorted in ascending order of measure(Aj)
2. For each point bi, 2≤i≤|Bj| o
3.   Rate-of-change(bi) = (bi - bi-1)/ bi // rate of change
4. Endfor

```

5. $\beta \leftarrow i$ with max Rate-of-change (b_i)
6. For each b_i , $1 \leq j \leq \beta$ do
7. OUTPUT CA-outliers $\leftarrow b_i$
8. Endfor

6 Experimental Validation

Experiments were performed on a Pentium-IV 3.2GHz computer with 2GB of main memory, and running Windows XP.

6.1 World Clock Dataset

The synthetic data set contains 9 attributes and 50,000 tuples generated from <http://www.timeanddate.com/worldclock/>. The original data set is free of any form of data noise, thus preventing the implicit noise in the original data set from interfering with the artificial noise introduced.

In order to evaluate the performance of ODDS at varying numbers of attribute outliers per tuple, we introduce x artificial attributes outliers to a random tuple in the data set. These attributes are assigned random values from their respective domains. The four datasets containing $x=1, 2, 3$, and 4 outliers per tuple are denoted X_1, X_2, X_3 and X_4 respectively. For example, X_2 has 2,500 CA-outliers distributed across 1,250 tuples, each containing 2 attribute outliers. We also generate a Mix3 dataset by randomly inserting 1 to 3 artificial attribute outliers to each randomly selected tuple.

The maximum Rate-of-change is the point where the outlier scores change significantly. Figure 3 shows the thresholds for individual attributes derived from the X_1 data set using the Rate-of-change on the O-measure scores in ODDS. These thresholds are used as the cut-off to determine the outliers (positives) from the non-outliers (negatives). Table 2 shows that an F-score of 100% is achieved for X_1 , indicating that the O-measure is effective in quantifying the extent of deviation in attribute outliers, and that the Rate-of-change accurately derives the optimal cut-off points. Subsequent experiments utilize the Rate-of-change factors as default selection for thresholds.

Table 2 shows the performance of ODDS at varying number of CA-outliers per tuple. With only 9 attributes, it is not surprising that the false-negatives escalate when tuples contain 4 or more CA-outliers per tuple. For data sets containing between 1 to 3 attribute outliers in each tuple, the outlier detection method can achieve an F-score of between 73% and 100%. We expect that real-world data set will contain a mixture of different number of attribute outliers in each tuple. For this, ODDS achieves an F-score of 88% for the Mix3 data set.

In reality, we do not know the number of attribute outliers that may be present in each tuple of a database. The ODDS approach systematically searches for CA-outliers, identifying tuples with only one outlier at the data subspaces of the highest degree k (i.e. complete tuple), and others at the subsequent lower degree projections.

The Mix3 data set is used to evaluate the performance of ODDS algorithm using O-measure, P-measure and Q-measure metrics respectively. On manual inspection, we find that these misses are in fact rare outlier countries which have been penalized by their low supports in the dataset (e.g. Chile, Iraq, Poland). The accuracy of ODDS converges across the projected relations of degree k , starting from $k=7$, with decreasing false negatives as the number of attribute outliers detected accumulate.

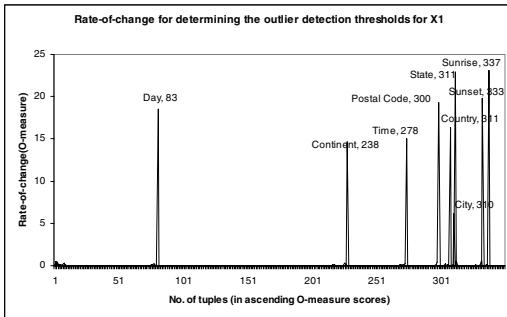


Fig. 3. Rate-of-change for individual attributes in X1

Figure 4 shows the F-score is between 70% to 80% with O-measure and Q-measure. For P-measure, the number of FN's is higher.

ODDS with O-measure and Q-measure perform consistently better than classifier-based methods using decision tree C4.5 [1, 5]. Its performance is also stable when the percentage of outlier noise increases. As the percentage of attribute outliers in the data set increases, the correlations between attributes decreases, thus affecting the accuracy of the correlation-based outlier detection approach.

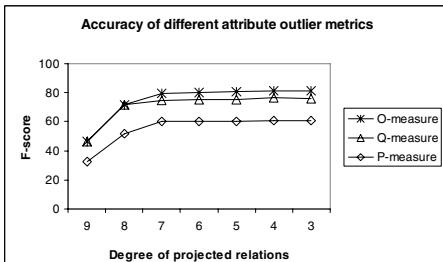


Fig. 4. Accuracy of ODDS metrics converges in data subspaces of lower degrees in Mix3

Table 2. Performance of ODDS at varying number of CA-outliers per tuple. Recall, Precision and F-score is in percentages.

	Recall	Precision	F-score
X1	100	100	100
X2	90	100	95
X3	63	99	73
X4	39	92	50
Mix3	79	99	88

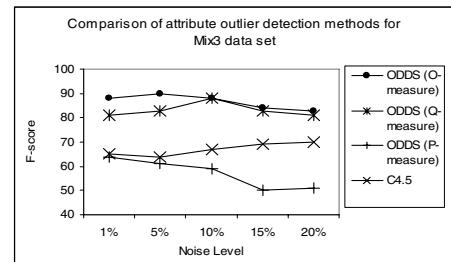


Fig. 5. Performance of ODDS compared with classifier-based attribute outlier detection

6.2 UniProt Dataset

The UniProt database (release 7.1) consists of 2,826,395 protein sequence records are collected from multiple sources of large-scale sequencing projects and is frequently accessed by the world-wide biological researchers for analysis and data mining [2]. UniProt/TrEMBL records are computationally annotated, thus the protein functions are predicted rather than verified experimentally, they contain a significant portion of mis-annotations or erroneous information [14, 15]. We apply ODDS on the UniProt database to identify discrepant annotations from 5 key attributes.

Table 3 shows that the protein name PN, gene name GN, synonym SY each contain more than 100,000 unique values. These large numbers suggest that the UniProt database is highly sparse. In fact, the naming of proteins and genes are often left to the discretion of the experimentalists who submit these sequences into the

Table 3. Performance of ODDS at varying number of CA-outliers per tuple

Attribute	Distinct values	Multiple values	Description	
OR	6	No	Organism source of the protein	
KW	898	Yes	Keywords subject reference for the protein	
GO	8486	Yes	Gene ontology controlled vocabulary of proteins' properties.	
PN	669,151	No	Proposed official name of protein	
SY	126,299	Yes	List of synonyms of the protein	

Table 4. CA-outliers detected in UniProt. Brackets contain number of affected records

CA-outliers detected at	OR	PN	KW	GO	SY
3-attribute combinations	27 (73)	45 (24)	56 (31)	17 (97)	18 (5)
4-attribute combinations	333 (553)	136 (6033)	276 (196)	378 (2196)	186 (124)
5-attribute combinations	195 (45)	40 (13)	57 (17)	308 (2365)	132 (56)
Accumulated	(671)	(6070)	(241)	(2365)	(185)

Table 5. Manual verification of GO CA-outliers detected in UniProt data set

Annotation	CA-outliers	TP	FP	Indet.
CA-outliers detected at 3-attribute combinations	17	6	5	6
CA-outliers detected at 4-attribute combinations	378	65	221	92
CA-outliers detected at 5-attribute combinations	308	31	136	141

database, hence, a large percentage of these names are rare but legitimate. Since we are not interested to detect these rare attribute values, we adopt the P-measure metric.

Table 4 shows the number of outliers detected for each attribute. We focus on the CA-outliers found in the GO dimension. The validities of these outliers are checked by biologists through manual verification. **True positive TP** indicates an uncommon association of the target attribute with the other attributes in the projected relation. **False positive FP** indicates that no peculiarity is found in the correlation behavior of the target attribute. **Indeterminable** means that further investigation is required.

The manual verification step largely depends on the knowledge level of the biologist and his decisive-ness. Table 5 shows that a large percentage (24%-46%) of the CA-outliers require further investigation because the biologist lacks the detail knowledge to justify if the annotation is erroneous or it is only exceptional. 27%-58% are false positives. 10%-24% of the gene ontology attribute outliers are confirmed result of erroneous annotations.

The experiment shows that ODDS can be used as a pre-step for cleaning protein annotations, subjected to further verification by an annotator. Obvious cases of erroneous annotations are found in the ODDS results. For example, 12 bacteria proteins (Q9Z5E4, Q6J5G7, among others) are associated with viral capsids which are protein coats for viral particles. Also, 5 eukaryote proteins (Q9BG87, Q4IJ15, among others) are oddly related to the reproduction of viruses.

7 Conclusion

Existing outlier detection methods focus primarily on class outliers; limited research has been conducted on attribute outliers. This paper presents a novel method called

ODDS that utilizes the correlations between attributes to identify attribute outliers. Rather than focusing on rare attribute values or rare classes, ODDS systematically searches for attribute points that exhibit alternative correlation behavior when compared to other attribute points in a data subspace. These local deviators which we refer to CA-outliers are dual-Experimental evaluation shows that ODDS can achieve F-score of up to 88% in synthetic data set and is practically applicable for detecting erroneous annotations in a protein database.

This paper focuses on the accuracy of the outlier detection approach. Two filtering strategies are used to improve the time efficiency of the ODDS algorithm where the enumeration of data subspaces is a major bottleneck. For future work, we strive to reduce the time complexity further. One strategy is to separate the data space into partitions of correlated subspaces in order to reduce the number of projections which are permuted.

Acknowledgments. Our thanks to the biologists - Mr. S.H. Tan, Mr. F. Clergeaud, and Mr. A.M. Khan.

References

1. Zhu, X., Wu, X.: Class Noise vs. Attribute Noise: A Quantitative Study of their Impacts. Artificial Intelligence Review, Vol. 22, No. 3 (2004) 177-210
2. Apweiler, R., Bairoch, A., Wu, C.H., *et al.*: UniProt: the Universal Protein Knowledgebase. Nucleic Acids Res. 32 (2004) 115-119
3. Barnett, V.: Outliers in Statistical Data. John Wiley and Sons, New York (1994)
4. Rousseeuw, P.J., Leroy, A.M.: Robust Regression and Outlier Detection. John Wiley (1987)
5. Choh, M.T.: Polishing Blemishes: Issues in Data Correction. IEEE Intelligent Systems, 19, issue 2 (2004) 34-39
6. He, Z., Xu, X., Deng, S.: Discovering cluster-based local outliers. Pattern Recognition Letters, vol. 24, Issue 9-10 (2003) 1641-1650
7. Jiang, M.F., Tseng, S.S., Su, C.M.: Two-phase clustering process for outliers detection. Pattern Recognition Letters, vol.22, Issue.6-7 (2001) 691-700
8. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: Lof: Identifying Density-based Local Outliers. ACM SIGMOD (2000) 93-104
9. Jin, W., Tung, A.K.H., Han, J.: Mining Top-n Local Outliers in Large Databases. SIGKDD (2001) 293-298
10. Papadimitriou, S., Kitagawa, H., Gibbons, P.B., Faloutsos, C.: LOCI: Fast Outlier Detection using the Local Correlation Integral. IEEE ICDE (2003) 315-326
11. Knorr, E.M., Ng, R.T., Tucakov, V.: Distance-based Outliers: Algorithms and Applications. VLDB Journal, 8 (2000) 237-253
12. Ramaswamy, S., Rastogi, R., Kyuseok, S.: Efficient Algorithm for Mining Outliers from Large Data Sets. ACM SIGMOD (2000) 427-438
13. Ren, D., Rahal, I., Perrizo, W., Scott, K.: A vertical distance-based outlier detection method with local pruning. ACM CIKM (2004) 279-284.
14. Gilks, W.R., Audit, B., De Angelis, D., *et al.*: Modeling the percolation of annotation errors in a data-base of protein sequences. Bioinformatics 18, 12, (2002) 1641-1649
15. Wieser, D., Kretschmann, E., Apweiler, R.: Filtering erroneous protein annotation. Bioinformatics, 20 (2004) i342-i347

An Efficient Histogram Method for Outlier Detection

Matthew Gebski and Raymond K. Wong

National ICT Australia, and University of New South Wales, Australia
`{francg,wong}@cse.unsw.edu.au`

Abstract. An important problem in database and data mining systems is the detection of outlying points. It is often the case that data observations exhibiting atypical properties are of more interest than those fitting common patterns. While anomaly and outlier detection have received considerable attention from the statistics community, these approaches are primarily focused on analysis of data sets containing relatively few and univariate observations. Recently, valuable approaches have been proposed to facilitate multidimensional analysis for larger data sets. Unfortunately, these approaches are often expensive and require numerous comparisons between each point and the remainder of the data.

We propose an approach using histograms for outlier detection. Sparse regions of the data are recognised and used for identifying points that are likely to be outliers. An extensive experimental evaluation demonstrates the efficiency of our approach under a number of circumstances with varying parameters on real world and synthetic data sets.

1 Introduction

With the increase in the size of databases, efficient techniques are required for analysis and interpretation of the stored data. An important data mining problem is outlier detection, where observations that deviate from the norm are identified. Tasks where outliers are valuable include credit card fraud analysis, determining adverse reactions to cancer treatments, or determining particularly profitable (or unprofitable) customers. Outlier detection involves examining points and discriminating based on some measure of “outlierness”.

Existing outlier detection algorithms for local outliers suffer due to the very large is the number of point-point comparisons required. Without optimisation, distance based algorithms need to test each point against all other points resulting in a time complexity of $O(n^2)$. Similarly, for local outlier algorithms running time is $O(n^2)$ as the k nearest neighbours for each point are used. The high cost of these algorithms makes analysis of large data sets difficult.

Recently, approaches have been proposed to improve the efficiency of distance based outlier calculations. These often involve randomisation and removing points from consideration once conditions can no longer be met. Unfortunately, these are not easily applied for detection of local outliers as the k nearest neighbours of each point would still be required.

We propose a novel approach in which histograms are used to find adjacent regions of consistent densities. By discarding points within these regions from consideration, we are able to significantly reduce the number of nearest neighbour calculations for local outliers. Two stages are used to first identify histogram buckets that are of interest and then identify points of interest. Following this, an optional reconsideration phase allows the removal of false positives.

2 Background

Hawkins [9] defines an outlier as “an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism”. We emphasise that the problem of outlier detection is different to that of detecting aberrant data. Univariate outlier analysis is a well studied problem in statistics [2] with two main types of outliers. Firstly, *parametric* or *distribution based* outliers, which are detected by examining how observations lie in relation to a known distribution function using a *discordancy test*. Often, it is difficult to ascertain the data distribution or an appropriate discordancy test may not exist. *Depth-based* outliers involve classifying points based on their ‘depth’ in relation to other observations. Points with smaller depths are more likely to be outliers over deeper points. Depth is calculated in a similar way to the construction of convex hulls and algorithms take $O(n^{d/2})$ time where d is the number of dimensions.

The first KDD approach for outlier detection was proposed by Knorr & Ng, using the distance between each point and each other point in the database [10, 11, 12, 13]. It was motivated by the difficulties in the application of discordancy tests and the cost of depth based techniques. Points are classified as *distance based* outliers if they are at least distance d from $p\%$ of the database. Knorr & Ng’s algorithm compares the distance from each point to the remainder of the population for this computation — this takes $O(n^2)$ time. Ramaswamy et al also used distance [16], ranking objects on the distance to their k nearest neighbours.

A randomised approach for computing distance based outliers was put forward by Bay & Schwabacher [4]. Experimental results show that this provides a significant speed increase. Kollios et al [14] use a density estimation stage for improving clustering algorithms, which is also applicable for distance based outliers and was shown to be accurate for two and three dimensions.

Distance based outliers provide no mechanism to vary the granularity of the distance measure over the database. For instance, with two normal distributions, a point may be 10 units and 1 standard deviation from the mean of the first distribution, while a second point may lie 2 units and 4 standard deviations from the other mean. It is now difficult to choose a cutoff that will result in the second point, but not the first, being classified as an outlier.

Density based outliers are classified depending on how the points are placed in local regions of the data. With OPTICS [5] and later LOF [6], Breunig et al determine the ‘outlierness’ of points by examination of a point’s k nearest neighbours (kNN). Points in neighbourhoods that are similar to nearby regions

are marked as regular, and those with neighbourhoods of varying density are classified as outliers. This allows regions of varying density to be processed individually, avoiding problems arising from data containing both numerous sparse and dense regions. Other local approaches include Papadimitriou et al's LOCI [15] which also examines the neighbourhood around each point.

We focus on local outliers, in particular those found by the LOF algorithm [6], due to the ability to detect types of outliers that are ungracefully handled by other approaches. Despite finding high quality outliers, the time required for the nearest neighbour queries may be expensive. As such, we are motivated to propose a more efficient approach for local outliers.

3 Approach

In this section, we present our approach for local outlier detection using histograms. We first identify data regions that can be discounted from containing outlying points. Following this, we discern points that seem to have a high degree of outlierness and consider these as *candidate outliers*. If we wish to completely eliminate false positives, these candidates are reconsidered in the context of the entire data set. Histograms (in particular, MinSkew) are chosen for their good accuracy for selectivity estimation. Furthermore, construction time is minimal [1], allowing a significant increase in performance over existing techniques.

3.1 Problem Description and Notation

We begin by defining a set of instances $D = \{p_1, p_2, \dots, p_n\}$, where each instance p_i is a point with d attributes, $p_i = \langle p_{i1}, p_{i2}, \dots, p_{id} \rangle$. When no ambiguity arises, we will use p to refer to an arbitrary point. For the remainder of this paper, we use the Euclidean distance metric with $d(p_i, p_j)$.

The aim is to determine the set of outlying points that satisfy a condition C relating outlying points to the remainder of the data set. More formally, we wish to find $\{p' \in D \mid C(p')\}$ where p' denotes a point that is an outlier. For instance, C may be true for points lying more than a certain distance from 90% of the entire data set. For the *local* outliers, $C(p)$ will be satisfied if a point is in a sufficiently sparse region bordering on a dense region.

For local outliers, we adjust the definitions of Breunig et al [6] for readability.

Definition 1. $k - \text{distance}(p)$ of a point p is the minimal distance d such that at least k points are within distance d from p .

Definition 2. *Reachability* of point p with respect to point o is defined as $\text{reachability}(o,p) = \max\{k - \text{distance}(o), \delta(o,p)\}$

Definition 3. *Local Reachability Density* of p is:

$$\text{lrd}(p) = \frac{k}{\sum_{o \in kNN(p)} \text{reachability}(o,p)}, \text{ where } kNN(p) \text{ refers to the } k \text{ nearest neighbours of } p.$$

Definition 4. Local Outlier Factor of a point p is: $LOF(p) = \frac{1}{k} \sum_{o \in kNN(p)} \frac{lrd(o)}{lrd(p)}$.

The local outlier factor is the average ratio between the local reachability density between each of p 's nearest neighbours and the local reachability density p . A point in a sparse region with neighbours in comparatively dense regions will subsequently have lower reachability densities than its neighbours. This results in a high outlierness for p ; if this is below a predetermined threshold, p will be marked as an outlier.

3.2 Histogram Based Outliers

Typically, histogram techniques iteratively divide existing buckets into smaller buckets to minimise a badness function [178]. Unless otherwise mentioned, we will use the MinSkew algorithm from Acharya et al [1]. This uses the sum of variance for each bucket as a badness function. That is, $\sum_1^n |B_i| s_i$ where $|B_i|$ is the number of points in bucket B_i , n is the total number of buckets and s_i is the statistical variance.

Proposition 1. If point p falls within bucket B with density $B.dens$ and p is more than $2r$ from any edge of B , p is not an outlier where d is the dimensionality of the data set (r is the radius of the hypersphere enclosing p 's k nearest neighbours).

If the histogram estimator is accurate, the area, a , of the circle enclosing all of p 's k nearest neighbours is $k/B.dens$. The distance to the furthest of these will be $\delta = \sqrt[d]{a/\pi}$. Denoting this neighbour as p'_i , the reachability distance for p is at most $d(p, p'_i) + d(p'_i, p'_j)$ where p'_j is the furthest of p'_i 's neighbours. If both p'_i and p'_j are contained within B , the maximum distance that p'_j can be from p'_i is δ . As such, p is not a density based outlier if $d(p, e_{B_i}) > \delta$, where $e_{B_i} \in \{\text{edges of } B_i\}$. This is illustrated in Figure 1a and Figure 1b.

Intuitively, we can consider that for selectivity estimation, we are attempting to create regions of uniform density. Points lying towards the centre of these regions are unlikely to be outliers. With Proposition 1, a large number of points can be removed from consideration as outliers. For data sets with large regions of relatively uniformly located data, this is particularly valuable (for instance the Forest Cover data set [3]).

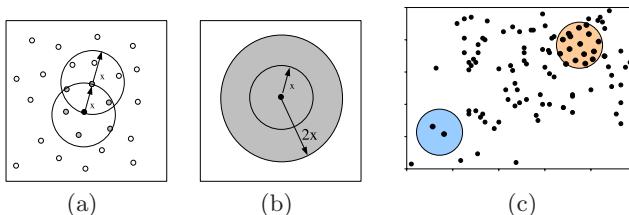


Fig. 1.

3.3 Histogram Refinement

Due to the MinSkew algorithm minimising variance during construction, uniform regions of the data may be divided. By attempting to minimise $\sum_{i=1}^m \sigma_{B_i}^2$, where m is the number of buckets in a partitioning P_m , it may be possible for a split to occur such that the accuracy of the partitioning is only trivially improved, $A(P_i) + \epsilon = A(P_{i+1})$.

Consider a bivariate data set with a point at each value between 1 and 500, i.e. (1,1), (1,2), ..., (500, 499), (500,500). For each attribute, we have $\sigma^2 = 20833.25$ and a split results in buckets with badness $\sigma_1^2 + \sigma_2^2 = 10416.5$. Despite this, we observe only a marginal increase in accuracy due to the structure. While we would not expect to observe this for the entire data set, it is easy to imagine local regions exhibiting such patterns. Using bucket merges, we are able to increase the number of points that can be discounted from being outliers.

Merging buckets is based on two factors. Firstly, the densities of the buckets so as not to unnecessarily mark points in regions of consistent densities as outliers. Secondly, we take into account the structure of the buckets; two adjacent buckets may in fact have similar densities, this may be because they each contain a cluster of points. During construction, the split was correct and it would be to our disadvantage if we were to merge the resulting bucket.

It is possible to restrict some merges during the histogram construction phase. Regions with uniformly placed points in this section is one such case. Other scenarios are addressed by consideration once the partitioning is constructed. The borders of adjacent buckets B_x and B_y are scanned, with regions of B_x consistent with B_y may be reassigned from B_x to B_y (or vice-versa). In cases where the buckets are very similar, the complete buckets will be merged.

In order to determine if two regions should be merged, we construct a sub-histogram within the buckets from the original histogram. To avoid confusion, we will use \mathcal{H} to represent the initial histogram and use \mathcal{H}' for the sub-histogram. Similarly, \mathcal{H}'_{B_p} will be used for indicating the bucket of \mathcal{H}' containing point p .

The merging process involves three phases for two adjacent buckets of \mathcal{H} , that is, \mathcal{H}_{B_i} and \mathcal{H}_{B_j} . Firstly, the construction of the equi-depth/equi-width sub-histograms, \mathcal{H}'_i for \mathcal{H}_{B_i} and \mathcal{H}'_j for \mathcal{H}_{B_j} . The second stage involves consideration of bordering cells $\mathcal{H}'_{B_x} \in \mathcal{H}'_i$ and $\mathcal{H}'_{B_y} \in \mathcal{H}'_j$. If the ratio of these densities are within an acceptable range $1 \pm \epsilon$ then the buckets are marked for merging. Finally, merge marked buckets and calculate densities of the resulting buckets. When merging buckets, the lowest of density values is chosen to represent the overall density. While this may result in some underestimation of the density of the combined bucket, the resulting sparse regions will lead to false positives which can be reconsidered later.

For selectivity estimation, if the relative error being used for accuracy $err = \frac{t - est}{t}$, where t represents the true selectivity and est represents the value estimated using the histogram, the difference between the error for the original and merged buckets will be $\pm \frac{Q_{area} \times \epsilon}{t}$ where Q_{area} is the area of the query for estimation.

Algorithm 1. Finds outliers using the histogram based approach

```

FIND-OUTLIERS-HISTOGRAM ( $D$ ):
1:  $\mathcal{H} \leftarrow \text{CONSTRUCT-HISTOGRAM}(D)$ 
2:  $\mathcal{H}' \leftarrow \text{MERGE-BUCKETS}(\mathcal{H})$ 
3:  $Candidates \leftarrow \emptyset, Outliers \leftarrow \emptyset$ 
4: for point  $p \in D$  do
5:    $\mathcal{H}'_{Bp} \leftarrow \text{FIND-BUCKET}(p, \mathcal{H}')$ 
6:   if  $\text{ISCONTAINED}(p, \mathcal{H}'_{Bp})$  then
7:      $p$  is not an outlier
8:   else
9:      $plrd \leftarrow 0$ 
10:     $olrd \leftarrow 0$ 
11:    for each  $\mathcal{H}'_{Bj}$  reachable for  $p$  do
12:      Update  $plrd$  based on  $\mathcal{H}'_{Bj}$ 
13:      Update  $olrd$  based on  $\mathcal{H}'_{Bj}$ 
14:      if  $\frac{plrd}{k \cdot olrd} > \tau'$  then
15:         $Candidates \cup p$ 
16:    for  $p \in Candidates$  do
17:      if  $\text{IS-OUTLIER}(p, D)$  then
18:         $Outliers \leftarrow Outliers \cup \{p\}$ 
19: return  $Outliers$ 

```

3.4 Refining Candidates

We can also use the histogram structure to identify points that seem particularly likely to be outliers or non outliers. Cells of the sub-histograms are used to approximate the densities surrounding each point. This allows estimation of the maximum and minimum LOF values for points and improve the identification of candidate outliers. The bounds allow us to better discriminate with regards to non outlying points in addition to helping us estimate LOF for outliers.

We first consider the density and size of both \mathcal{H}_{Bp} and \mathcal{H}'_{Bp} . If a point is on the edge of the bucket of size $\kappa > k$, the local reachability density of p cannot be greater than $\text{diag}(Bp)^{-1}$, where $\text{diag}(Bp)$ is the length of the diagonal of Bp . In the most skewed scenario, the reachability distance for p must be less than that of the diagonal of the bucket. When closer to the centre, this is likely to be much smaller. For any point, the expected value for two dimensional data of $lrd(p)$ (only considering \mathcal{H}'_{Bp}) will be $(\sqrt{\frac{k \mathcal{H}'_{Bp}.density}{\pi}})^{-1}$. As the dimensionality increases, the expression should be altered to consider the radius of the appropriate hypersphere. For points near the bucket edge, we also consider the density of the relevant buckets adjacent to \mathcal{H}'_{Bp} . A pessimistic approach is taken, such that the estimated lrd will be lower than the true value. While this may increase the number of false positives, it mitigates the number of missed outliers. Of the buckets under consideration, we use the lowest density value.

The second stage is to estimate the range of possible *LOF* values for p . Initially, it is assumed that the positions of neighbouring points place p in a sparse region of the data, allowing for the lower *LOF* estimate. Following this, the highest density regions nearby are examined and are used for the upper *LOF* estimate. It is important to note that this estimate is only performed for p . Any of the other points that lie in the regions close to p are considered independently. The estimates can then be used for determining the densities of the point and points in nearby regions and then the range into which the *LOF* score will fall. Points which are highly unlikely to be outliers are immediately discarded. A threshold value, T , is used for choosing points that should be removed from consideration. This value can be modified with ease, however it is rarely useful to keep points that will have a *LOF* of 1.0 or less.

Algorithm 1 shows the approach for histogram based outlier detection. τ' is the threshold above which we consider a point a candidate outlier. The method *Is – Outlier* in line 18 returns true if the point is an outlier in the context of the whole data set using the *LOF* algorithm.

3.5 Accuracy

Consider a bucket with a heavily skewed distribution such as depicted in Figure 1c. The estimated density of points in the upper right region will be greater than the true density while for points in the lower left region, the estimated density will be lower. This decreases the likelihood upper right points being classified as outliers while increasing the likelihood for those in the lower left.

To assist us in considering the error induced by the approach, we define two terms that relate to the estimate of density surrounding a given point:

Definition 5. *Relative overestimate* for point p is the ratio of the densities between the region surrounding p and the regions surrounding its neighbours estimated by the histogram to be higher than the true ratio. That is, $estlrd(p)/estlrd(neighbour) > truelrd(p)/truelrd(neighbour)$.

As histogram construction takes place with constraints on the number of buckets permitted, points in sparse regions may be placed in comparatively dense buckets. The overestimate may be due to either that the histogram is accurate for the density surrounding a point's neighbours, but not for the point itself ($estlrd(neighbours) = truelrd(neighbours)$ and $estlrd(p) > truelrd(p)$), or the density is correct for p , but is underestimated for p 's neighbours ($estlrd(p) = truelrd(p)$ and $estlrd(neighbours) < truelrd(p)$).

Definition 6. *Relative underestimate* is the ratio of the densities between the region surrounding p and the regions surrounding its neighbours estimated by the histogram to be lower than the true ratio. That is, $estlrd(p)/estlrd(neighbour) < lrd(p)/lrd(neighbour)$.

As with the overestimation, relative underestimation is a result of the constraints on the maximum number of buckets. However, points in sparse regions are in

buckets with a higher density than the density surrounding each point. There are two cases to consider, firstly, $\text{estlrd(neighbours)} = \text{truelrd(neighbours)}$ and $\text{estlrd}(p) < \text{truelrd}(p)$. Or secondly, where $\text{estlrd}(p) = \text{truelrd}(p)$ and then $\text{estlrd(neighbours)} > \text{truelrd}(p)$.

Proposition 2. A relative underestimate for a non-outlying point p may lead to p being misidentified as an outlier if $\frac{\text{lrd}(p) - \text{estlrd}(p)}{k} > \tau - \gamma$.

Consider a point p with a local outlier factor of γ and assume that $\gamma < \tau$ where τ is the threshold above which points are considered outliers. Recall that when densities are computed from the full data set, $\gamma = \frac{1}{k} \sum_{o \in kNN(p)} \frac{\text{lrd}(p)}{\text{lrd}(o)}$. Now, if there is a relative underestimate for the region surrounding p with $\text{estlrd}(p) < \text{lrd}(p)$ and $\text{estlrd}(o) = \text{lrd}(o)$ then

$$\gamma > \frac{1}{k} \sum_{o \in kNN(p)} \frac{\text{lrd}(o)}{\text{estlrd}(p)}.$$

Now if $\frac{\text{lrd}(p) - \text{estlrd}(p)}{k} > \tau - \gamma$, then p will be erroneously be marked as an outlier.

The opposite of the previous case is:

Proposition 3. A relative underestimate for a outlying point p may lead to p being incorrectly identified as a non-outlier. A point will be no longer be marked as an outlier if $\frac{\text{estlrd}(p) - \text{lrd}(p)}{k} > \gamma - \tau$.

4 Experimental Evaluation

Two sets of experiments are run; the first set involve synthetic data used to examine how different distributions and parameters affect the approach. Two synthetic sets are used: data drawn from the normal and exponential distributions and is iid. We also examine the Forest Cover [3] data set; this represents

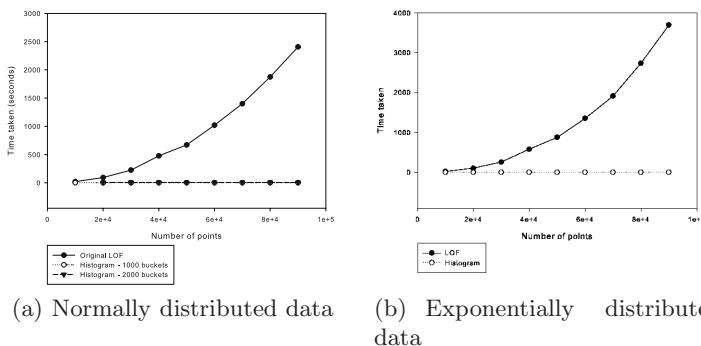


Fig. 2. Running times for Histogram & LOF algorithms for normally and exponentially distributed data

observations of forest information taken by the United States Forest Service. The data contains information such as elevation, aspect and slope for forest ‘cells’. Nominal data such as soil type were excluded.

Of course, if the distribution parameters are known, we would expect no outliers simply extreme values. In our experiments, we make no assumptions about the distribution from which the data has been drawn and LOF and local densities for the discrimination of outlying points.

4.1 Results

For Figure 2, two constraints are used for the buckets, the first is 1,000 (10% for the smallest size set moving towards 1% for the largest) and the second 2,000. For 1,000 buckets, slightly more than 2 seconds are required for the 10,000 point set and marginally less than 3 (2.91 seconds) for the 100,000 point set. Approximately 7.6 seconds are needed for 20,000 points with 2,000 buckets,

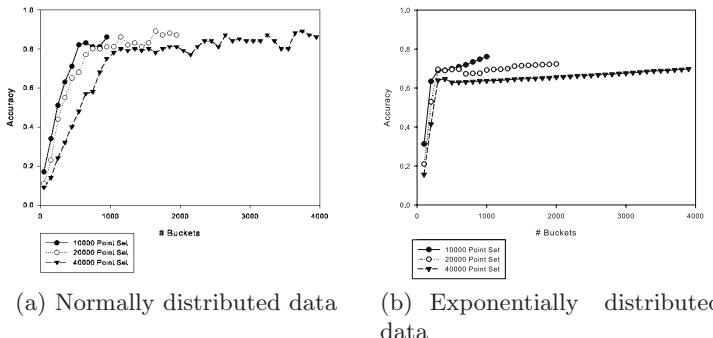


Fig. 3. Effect of varying the number of buckets on accuracy for normally and exponentially distributed data

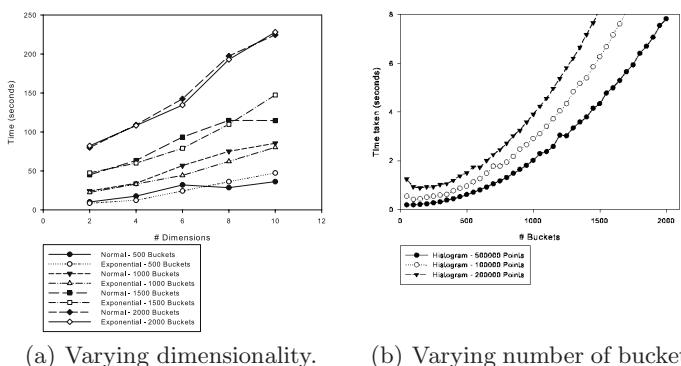


Fig. 4. Effect of varying dimensionality and number of buckets on running time

peaking at 8.3 seconds for 100,000 points with 2,000 buckets. Figure 2b shows the impact on time when the number of buckets is varied. When keeping the number of buckets constant, there is only a slight increase in running time compared to the increase in population size. Allowing the number of buckets to increase along with the number of points, there is an increase in the time taken. This is a result of the increased number of points close to the border of a cell.

Figure 3a shows the accuracy as we vary the number of buckets. The data set sizes are 10,000, 20,000 and 40,000. There is a sharp increase as the number of buckets approaches 500. The accuracy then plateaus at approximately 80% accuracy with marginal increases in accuracy resulting from the increases in the number of buckets. Additionally, when doubling the size of the data set, the number of buckets needed to achieve the accuracy of the smaller set is small. This suggests that as the histogram can model the shape of the data, the number of buckets required decreases proportionally against the size of the data set.

The results for varying data dimensionality are shown in Figure 2a. As the dimensionality increases that performance is slightly impacted. The accuracy is still competitive for data of lower dimensionality. The sets used for these experiments consist of 200,000 points and for the most part, outliers are found in well under one minute. Even for the higher dimensional data, the performance is still superior to that of LOF running on two dimensional data.

For exponentially distributed data our approach again outperforms LOF (Figure 2b). Only a couple of seconds are required for 100,000 instances, which is consistent with the normally distributed data results. The time for LOF is very similar which is what we would expect; the number of nearest neighbour queries required does not change with the structure of the data.

Because of the structure of the exponential data, we observe a slight decrease in the accuracy of our approach in Figure 3b. Again, there is a dramatic increase in the quality as the number of buckets increases from 0 until approximately 400–500. The lower accuracy is a result of the structure of the data. The exponential data contains a diagonal ‘cutoff’ between the regions where points lie and the remaining empty space. As rectangular buckets are used, each bucket containing points in this region typically contains a large empty portion. Both the performance and accuracy are affected in a consistent manner for the exponentially and normally distributed data sets.

A number of subsets were constructed from the UCI KDD Forest Cover data set [3]. Only unique values were used (spikes in the data may result in the k nearest neighbours of a point having the same value as the point). The first subset contains the attribute “Horizontal_Distance_To_Hydrology” (HDH) as well as “Vertical_Distance_To_Hydrology” (VDH) this consists of approximately 66,000 unique instances. The second contains “Slope”, “Aspect”, in addition to HDH and VDH with approximately 545,000 unique instances. Due to the large size, only our approach was run for these data sets.

Figure 5 shows the performance of our approach as the size of the data set is increased. For the two dimensional data set, the number of buckets was set at 10% of the number of points present. For the four dimensional set, the number of

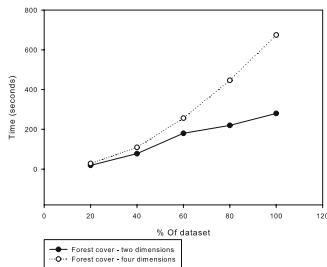


Fig. 5. Running times for the Forest Cover data

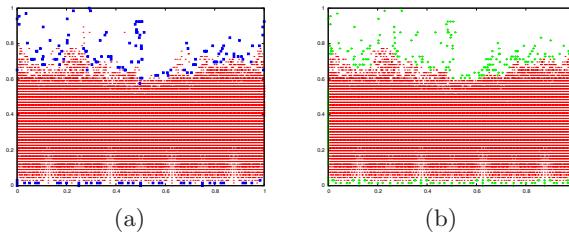


Fig. 6. Outliers found by Histogram and LOF approaches

buckets was set at 2%. This also allows us to appreciate the additional increase in set size and dimensionality. Figure 6 shows the structure of a cross section of the data containing the attributes “Aspect” and “Slope”. The large regular regions can easily be modelled with the histograms with only a small impact on the quality of the outliers, leading to significant performance gain.

5 Conclusions

We have examined an approach for local outlier detection using histograms to efficiently approximate densities rather than explicit computation using nearest neighbours. The time taken for existing techniques is considerable; our approach allows outliers to be found much faster with only a small decrease in accuracy. A number of steps are used, the first of which involves refinement of the histogram buckets. This is followed by removal of points located in the centre of large buckets. The third step is to examine each point and estimate the density by examining the surrounding region in the histogram. If the maximum Local Outlier Factor score that a point can have is below a specified threshold, we immediately remove the point from consideration.

References

1. S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *SIGMOD*, pages 13–24. ACM Press, 1999.
2. V. Barnett and T. Lewis. Outliers in statistical data. John Wiley, 1994.

3. S. D. Bay, D. Kibler, M. J. Pazzani, and P. Smyth. The UCI KDD archive of large data sets for data mining research and experimentation. *Information Processing Society of Japan Magazine*, 42(5):462–466, 2001.
4. S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos, editors, *KDD*, pages 29–38. ACM, 2003.
5. M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Optics-of: Identifying local outliers. In J. M. Zytkow and J. Rauch, editors, *PKDD*, pages 262–270, 1999.
6. M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *SIGMOD*, pages 93–104. ACM, 2000.
7. A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is good: dependency-based histogram synopses for high-dimensional data. In *SIGMOD*, pages 199–210. ACM Press, 2001.
8. D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *SIGMOD*, pages 463–474. ACM Press, 2000.
9. D. Hawkins. Identification of outliers. Chapman and Hall, London, 1980.
10. E. M. Knorr and R. T. Ng. A unified approach for mining outliers. In *CASCON*, page 11. IBM, 1997.
11. E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, pages 392–403. Morgan Kaufmann, 1998.
12. E. M. Knorr and R. T. Ng. Finding intensional knowledge of distance-based outliers. In *VLDB*, pages 211–222. Morgan Kaufmann, 1999.
13. E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. *VLDB J.*, 8(3-4):237–253, 2000.
14. G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold. Efficient biased sampling for approximate clustering and outlier detection in large datasets, 2003.
15. S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *ICDE*, pages 315–326, 2003.
16. S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *SIGMOD*, pages 427–438. ACM Press, 2000.

Efficient k -Anonymization Using Clustering Techniques*

Ji-Won Byun¹, Ashish Kamra², Elisa Bertino¹, and Ninghui Li¹

¹ CERIAS and Computer Science, Purdue University

{byunj, bertino, ninghui}@cs.purdue.edu

² CERIAS and Electrical and Computer Engineering, Purdue University

akamra@purdue.edu

Abstract. k -anonymization techniques have been the focus of intense research in the last few years. An important requirement for such techniques is to ensure anonymization of data while at the same time minimizing the information loss resulting from data modifications. In this paper we propose an approach that uses the idea of clustering to minimize information loss and thus ensure good data quality. The key observation here is that data records that are naturally similar to each other should be part of the same equivalence class. We thus formulate a specific clustering problem, referred to as *k -member clustering problem*. We prove that this problem is NP-hard and present a greedy heuristic, the complexity of which is in $O(n^2)$. As part of our approach we develop a suitable metric to estimate the information loss introduced by generalizations, which works for both numeric and categorical data.

1 Introduction

A recent approach addressing data privacy relies on the notion of *k -anonymity* [1][13]. In this approach, data privacy is guaranteed by ensuring that any record in the released data is indistinguishable from at least $(k - 1)$ other records with respect to a set of attributes called the *quasi-identifier*. Although the idea of k -anonymity is conceptually straightforward, the computational complexity of finding an optimal solution for the k -anonymity problem has been shown to be NP-hard, even when one considers only cell suppression [19]. The k -anonymity problem has recently drawn considerable interest from research community, and a number of algorithms have been proposed [3][6][7][8][12]. Current solutions, however, suffer from high information loss mainly due to reliance on pre-defined generalization hierarchies [4][6][7][12] or total order [3][8] imposed on each attribute domain. We discuss these algorithms more in detail in Section 2.

The main goal of our work is to develop a new k -anonymization approach that addresses such limitations. The key idea underlying our approach is that the k -anonymization problem can be viewed as a clustering problem. Intuitively, the k -anonymity requirement can be naturally transformed into a clustering

* This material is based upon work supported by the National Science Foundation under Grant No. 0430274 and the sponsors of CERIAS.

problem where we want to find a set of clusters (i.e., equivalence classes), each of which contains at least k records. In order to maximize data quality, we also want the records in a cluster to be as similar to each other as possible. This ensures that less distortion is required when the records in a cluster are modified to have the same quasi-identifier value. We thus formulate a specific clustering problem, which we call *k -member clustering problem*. We prove that this problem is NP-hard and present a greedy algorithm which runs in time $O(n^2)$. Although our approach does not rely on generalization hierarchies, if there exist some natural relations among the values in a domain, our algorithm can incorporate such information to find more desirable solutions. We note that while many quality metrics have been proposed for the hierarchy-based generalization, a metric that precisely measures the information loss introduced by the hierarchy-free generalization has not yet been introduced. For this reason, we define a data quality metric for the hierarchy-free generalization, which we call *information loss metric*. We also show that with a small modification, our algorithm is able to reduce classification errors effectively.

The remainder of this paper is organized as follows. We review the basic concepts of the k -anonymity model and survey existing techniques in Section 2. We formally define the problem of k -anonymization as a clustering problem and introduce our approach in Section 3. Then we evaluate our approach based on the experimental results in Section 4. We conclude our discussion in Section 5.

2 Preliminaries

2.1 Basic Concepts

The k -anonymity model assumes that person-specific data are stored in a table (or a relation) of columns (or attributes) and rows (or records). The process of anonymizing such a table starts with removing all the explicit identifiers, such as name and SSN, from the table. However, even though a table is free of explicit identifiers, some of the remaining attributes in combination could be specific enough to identify individuals if the values are already known to the public. For example, as shown by Sweeney [13], most individuals in the United States can be uniquely identified by a set of attributes such as {ZIP, gender, date of birth}. Thus, even if each attribute alone is not specific enough to identify individuals, a group of certain attributes together may identify a particular individual. The set of such attributes is called *quasi-identifier*.

The main objective of the k -anonymity model is thus to transform a table so that no one can make high-probability associations between records in the table and the corresponding entities. In order to achieve this goal, the k -anonymity model requires that any record in a table be indistinguishable from at least $(k - 1)$ other records with respect to the pre-determined quasi-identifier. A group of records that are indistinguishable to each other is often referred to as an *equivalence class*. By enforcing the k -anonymity requirement, it is guaranteed that even though an adversary knows that a k -anonymous table contains the record of a particular individual and also knows some of the quasi-identifier

ZIP	Gender	Age	Diagnosis
47918	Male	35	Cancer
47906	Male	33	HIV+
47918	Male	36	Flu
47916	Female	39	Obesity
47907	Male	33	Cancer
47906	Female	33	Flu

Fig. 1. Patient Table

ZIP	Gender	Age	Diagnosis
4791*	Person	[35-39]	Cancer
4790*	Person	[30-34]	HIV+
4791*	Person	[35-39]	Flu
4791*	Person	[35-39]	Obesity
4790*	Person	[30-34]	Cancer
4790*	Person	[30-34]	Flu

Fig. 2. 3-anonymous Patient table

attribute values of the individual, he/she cannot determine which record in the table corresponds to the individual with a probability greater than $1/k$. For example, a 3-anonymous version of the table in Fig. 1 is shown in Fig. 2

2.2 Existing Techniques

The k -anonymity requirement is typically enforced through *generalization*, where real values are replaced with “less specific but semantically consistent values” [13]. Given a domain, there are various ways to generalize the values in the domain. Typically, numeric values are generalized into intervals (e.g., [12–19]), and categorical values are generalized into a set of distinct values (e.g., {USA, Canada}) or a single value that represents such a set (e.g., North-America).

Various generalization strategies have been proposed. In [7,11,12], a non-overlapping generalization-hierarchy is first defined for each attribute of quasi-identifier. Then an algorithm tries to find an optimal (or good) solution which is allowed by such generalization hierarchies. Note that in these schemes, if a lower level domain needs to be generalized to a higher level domain, all the values in the lower domain are generalized to the higher domain. This restriction could be a significant drawback in that it may lead to relatively high data distortion due to unnecessary generalization. The algorithms in [4,6], on the other hand, allow values from different domain levels to be combined to represent a generalization. Although this leads to much more flexible generalization, possible generalizations are still limited by the imposed generalization hierarchies.

Recently, some schemes that do not rely on generalization hierarchies [3,8] have been proposed. For instance, LeFevre et al. [8] transform the k -anonymity problem into a partitioning problem. Specifically, their approach consists of the following two steps. The first step is to find a partitioning of the d -dimensional space, where d is the number of attributes in the quasi-identifier, such that each partition contains at least k records. Then the records in each partition are generalized so that they all share the same quasi-identifier value. Although shown to be efficient, these approaches also have a disadvantage that it requires a total order for each attribute domain. This makes it impractical in most cases involving categorical data which have no meaningful order.

3 Anonymization and Clustering

The key idea underlying our approach is that the k -anonymization problem can be viewed as a clustering problem. Clustering is the problem of partitioning a set of objects into groups such that objects in the same group are more similar to each other than objects in other groups with respect to some defined similarity criteria [5]. Intuitively, an optimal solution of the k -anonymization problem is indeed a set of equivalence classes such that records in the same equivalence class are very similar to each other, thus requiring a minimum generalization.

3.1 k -Anonymization as a Clustering Problem

Typical clustering problems require that a specific number of clusters be found in solutions. However, the k -anonymity problem does not have a constraint on the number of clusters; instead, it requires that each cluster contains at least k records. Thus, we pose the k -anonymity problem as a clustering problem, referred to as *k -member clustering problem*.

Definition 1. (k -member clustering problem) The k -member clustering problem is to find a set of clusters from a given set of n records such that each cluster contains at least k ($k \leq n$) data points and that the sum of all intra-cluster distances is minimized. Formally, let \mathcal{S} be a set of n records and k the specified anonymization parameter. Then the optimal solution of the k -clustering problem is a set of clusters $\mathcal{E} = \{e_1, \dots, e_m\}$ such that:

1. $\forall i \neq j \in \{1, \dots, m\}, e_i \cap e_j = \emptyset$,
2. $\bigcup_{i=1, \dots, m} e_i = \mathcal{S}$,
3. $\forall e_i \in \mathcal{E}, |e_i| \geq k$, and
4. $\sum_{\ell=1, \dots, m} |e_\ell| \cdot \text{MAX}_{i,j=1, \dots, |e_\ell|} \Delta(p_{(\ell,i)}, p_{(\ell,j)})$ is minimized.

Here $|e|$ is the size of cluster e , $p_{(\ell,i)}$ represents the i -th data point in cluster e_ℓ , and $\Delta(x, y)$ is the distance between two data points x and y . \square

Note that in Definition 1, we consider the sum of all intra-cluster distances, where an intra-cluster distance of a cluster is defined as the maximum distance between any two data points in the cluster (i.e., the diameter of the cluster). As we describe in the following section, this sum captures the total information loss, which is the amount of data distortion that generalizations introduce to the entire table.

3.2 Distance and Cost Metrics

At the heart of every clustering problem are the distance functions that measure the dissimilarities among data points and the cost function which the clustering problem tries to minimize. The distance functions are usually determined by the type of data (i.e., numeric or categorical) being clustered, while the cost function is defined by the specific objective of the clustering problem. In this section, we

describe our distance and cost functions which have been specifically tailored for the k -anonymization problem.

As previously discussed, a distance function in a clustering problem measures how dissimilar two data points are. As the data we consider in the k -anonymity problem are person-specific records that typically consist of both numeric and categorical attributes, we need a distance function that can handle both types of data at the same time.

For a numeric attribute, the difference between two values (e.g., $|x - y|$) naturally describes the dissimilarity (i.e., distance) of the values. This measure is also suitable for the k -anonymization problem. To see this, recall that when records in the same equivalence class are generalized, the generalized quasi-identifier must subsume all the attribute values in the equivalence class. That is, the generalization of two values x and y in a numeric attribute is typically represented as a range $[x, y]$, provided that $x < y$. Thus, the difference captures the amount of distortion caused by the generalization process to the respective attribute (i.e., the length of the range).

Definition 2. (Distance between two numeric values) Let \mathcal{D} be a finite numeric domain. Then the normalized distance between two values $v_i, v_j \in \mathcal{D}$ is defined as:

$$\delta_N(v_1, v_2) = |v_1 - v_2| / |\mathcal{D}|,$$

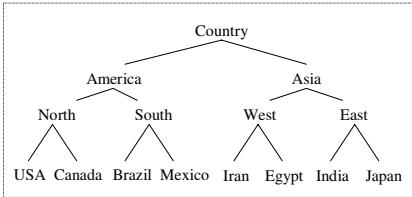
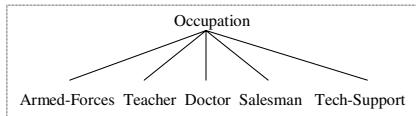
where $|\mathcal{D}|$ is the domain size measured by the difference between the maximum and minimum values in \mathcal{D} . \square

For categorical attributes, however, the difference is no longer applicable as most of the categorical domains cannot be enumerated in any specific order. The most straightforward solution is to assume that every value in such a domain is equally different to each other; e.g., the distance of two values is 0 if they are the same, and 1 if different. However, some domains may have some semantic relationships among the values. In such domains, it is desirable to define the distance functions based on the existing relationships. Such relationships can be easily captured in a *taxonomy tree*¹. We assume that a taxonomy tree of a domain is a balanced tree of which the leaf nodes represent all the distinct values in the domain. For example, Fig. 3 illustrates a natural taxonomy tree for the *Country* attribute. However, for some attributes such as *Occupation*, there may not exist any semantic relationship which can help in classifying the domain values. For such domains, all the values are classified under a common value as in Fig. 4. We now define the distance function for categorical values as follows:

Definition 3. (Distance between two categorical values) Let \mathcal{D} be a categorical domain and $\mathcal{T}_{\mathcal{D}}$ be a taxonomy tree defined for \mathcal{D} . The normalized distance between two values $v_i, v_j \in \mathcal{D}$ is defined as:

$$\delta_C(v_1, v_2) = H(\Lambda(v_i, v_j)) / H(\mathcal{T}_{\mathcal{D}}),$$

¹ Taxonomy tree can be considered similar to generalization hierarchy introduced in [7][11][12]. However, we treat taxonomy tree not as a restriction, but a user's preference.

**Fig. 3.** Taxonomy tree of *Country***Fig. 4.** Taxonomy tree of *Occupation*

where $\Lambda(x, y)$ is the subtree rooted at the lowest common ancestor of x and y , and $H(R)$ represents the height of tree \mathcal{T} . \square

Example 1. Consider attribute *Country* and its taxonomy tree in Fig. 3. The distance between *India* and *USA* is $3/3 = 1$, while the distance between *India* and *Iran* is $2/3 = 0.66$. On the other hand, for attribute *Occupation* and its taxonomy tree in Fig. 4, which goes up only one level, the distance between any two values is always 1.

Combining the distance functions for both numeric and categorical domains, we define the distance between two records as follows:

Definition 4. (Distance between two records) Let $\mathcal{Q}_T = \{N_1, \dots, N_m, C_1, \dots, C_n\}$ be the quasi-identifier of table T , where $N_i(i = 1, \dots, m)$ is an attribute with a numeric domain and $C_j(j = 1, \dots, n)$ is an attribute with a categorical domain. The distance of two records $r_1, r_2 \in T$ is defined as:

$$\Delta(r_1, r_2) = \sum_{i=1, \dots, m} \delta_N(r_1[N_i], r_2[N_i]) + \sum_{j=1, \dots, n} \delta_C(r_1[C_j], r_2[C_j]),$$

where $r_i[A]$ represents the value of attribute A in r_i , and δ_N and δ_C are the distance functions defined in Definitions 2 and 3, respectively. \square

Now we discuss the cost function which the k -members clustering problem tries to minimize. As the ultimate goal of our clustering problem is the k -anonymization of data, we formulate the cost function to represent the amount of distortion (i.e., information loss) caused by the generalization process. Recall that, records in each cluster are generalized to share the same quasi-identifier value that represents every original quasi-identifier value in the cluster. We assume that the numeric values are generalized into a range $[\min, \max]$ [2] and categorical values into a set that unions all distinct values in the cluster [3]. With these assumptions, we define a metric, referred to as *Information Loss* metric (IL), that measures the amount of distortion introduced by the generalization process to a cluster.

Definition 5. (Information loss) Let $e = \{r_1, \dots, r_k\}$ be a cluster (i.e., equivalence class) where the quasi-identifier consists of numeric attributes N_1, \dots, N_m and categorical attributes C_1, \dots, C_n . Let \mathcal{T}_{C_i} be the taxonomy tree defined for

the domain of categorical attribute C_i . Let MIN_{N_i} and MAX_{N_i} be the min and max values in e with respect to attribute N_i , and let \cup_{C_i} be the union set of values in e with respect to attribute C_i . Then the amount of information loss occurred by generalizing e , denoted by $IL(e)$, is defined as:

$$IL(e) = |e| \cdot \left(\sum_{i=1, \dots, m} \frac{(MAX_{N_i} - MIN_{N_i})}{|N_i|} + \sum_{j=1, \dots, n} \frac{H(\Lambda(\cup_{C_j}))}{H(T_{C_j})} \right)$$

where $|e|$ is the number of records in e , $|N|$ represents the size of numeric domain N , $\Lambda(\cup_{C_j})$ is the subtree rooted at the lowest common ancestor of every value in \cup_{C_j} , and $H(\mathcal{T})$ is the height of taxonomy tree \mathcal{T} . \square

Using the definition above, the total information loss of the anonymized table is defined as follows:

Definition 6. (Total information loss) Let \mathcal{E} be the set of all equivalence classes in the anonymized table \mathcal{AT} . Then the amount of total information loss of \mathcal{AT} is defined as:

$$\text{Total-IL}(\mathcal{AT}) = \sum_{e \in \mathcal{E}} IL(e).$$

 \square

Recall that the cost function of the k -members problem is the sum of all intra-cluster distances, where an intra-cluster distance of a cluster is defined as the maximum distance between any two data points in the cluster. Now, if we consider how records in each cluster are generalized, minimizing the total information loss of the anonymized table intuitively minimizes the cost function for the k -members clustering problem as well. Therefore, the cost function that we want to minimize in the clustering process is Total-IL.

3.3 Anonymization Algorithm

Armed with the distance and cost functions, we are now ready to discuss the k -member clustering algorithm. As in most clustering problems, an exhaustive search for an optimal solution of the k -member clustering is potentially exponential. In order to precisely characterize the computational complexity of the problem, we define the k -member clustering problem as a decision problem as follows.

Definition 7. (k -member clustering decision problem) Given n records, is there a clustering scheme $\mathcal{E} = \{e_1, \dots, e_\ell\}$ such that

1. $|e_i| \geq k$, $1 < k \leq n$: the size of each cluster is greater than or equal to a positive integer k , and
2. $\sum_{i=1, \dots, \ell} IL(e_i) < c$, $c > 0$: the Total-IL of the clustering scheme is less than a positive constant c .

 \square

Theorem 1. *The k -member clustering decision problem is NP-complete.*

Proof. That the k -member clustering decision problem is in NP follows from the observation that if such a clustering scheme is given, verifying that it satisfies the two conditions in Definition 7 can be done in polynomial time.

In [1], Aggarwal et al. proved that optimal k -anonymity by suppression is NP-hard, using a reduction from the EDGE PARTITION INTO TRIANGLES problem. In the reduction, the table to be k -anonymized consists of n records; each record has m attributes, and each attribute takes a value from $\{0, 1, 2\}$. The k -anonymization technique used is to suppress some cells in the table. Aggarwal et al. showed that determining whether there exists a 3-anonymization of a table by suppressing certain number of cells is NP-hard.

We observe that the problem in [1] is a special case of the k -member clustering problem where each attribute is categorical and has a flat taxonomy tree. It thus follows that the k -member clustering problem is also NP-hard. When each attribute has a flat taxonomy tree, the only way to generalize a cell is to the root of the flat taxonomy tree, and this is equivalent to suppressing the cell. Given such a database, the information loss of each record in any generalization is the same as the number of cells in the record that differ from any other record in the equivalent class, which equals the number of cells to be suppressed. Therefore, there exists a k -anonymization with total information loss no more than t if and only if there exists a k -anonymization that suppresses at most t cells. \square

Faced with the hardness of the problem, we propose a simple and efficient algorithm that finds a solution in a greedy manner. The idea is as follows. Given a set of n records, we first randomly pick a record r_i and make it as a cluster e_1 . Then we choose a record r_j that makes $IL(e_1 \cup \{r_j\})$ minimal. We repeat this until $|e_1| = k$. When $|e_1|$ reaches k , we choose a record that is furthest from r_i and repeat the clustering process until there are less than k records left. We then iterate over these leftover records and insert each record into a cluster with respect to which the increment of the information loss is minimal. We provide the core of our greedy k -member clustering algorithm, leaving out some trivial functions, in Figure 5.

Theorem 2. *Let n be the total number of input records and k be the specified anonymity parameter. Every cluster that the greedy k -member clustering algorithm finds has at least k records, but no more than $2k - 1$ records.*

Proof. Let \mathcal{S} be the set of input records. As the algorithm finds a cluster with exactly k records as long as the number of remaining records is equal to or greater than k , every cluster contains at least k records. If there remain less than k records, these leftover records are distributed to the clusters that are already found. That is, in the worst case, $k - 1$ remaining records are added to a single cluster which already contains k records. Therefore, the maximum size of a cluster is $2k - 1$. \square

Theorem 3. *Let n be the total number of input records and k be the specified anonymity parameter. The time complexity of the greedy k -member clustering algorithm is in $O(n^2)$.*

```

Function greedy_k_member_clustering ( $S, k$ )
Input: a set of records  $S$  and a threshold value  $k$ .
Output: a set of clusters each of which contains at least  $k$ 
records.
1. if(  $|S| \leq k$  )
2.   return  $S$ ;
3. end if;
4. result =  $\emptyset$ ;  $r$  = a randomly picked record from  $S$ ;
5. while(  $|S| \geq k$  )
6.    $r$  = the furthest record from  $r$ ;
7.    $S = S - \{r\}$ ;
8.    $c = \{r\}$ ;
9.   while(  $|c| < k$  )
10.     $r = find\_best\_record(S, c)$ ;
11.     $S = S - \{r\}$ ;
12.     $c = c \cup \{r\}$ ;
13.   end while;
14.   result = result  $\cup$  { $c$ };
15. end while;
16. while(  $|S| \neq 0$  )
17.    $r$  = a randomly picked record from  $S$ ;
18.    $S = S - \{r\}$ ;
19.    $c = find\_best\_cluster(result, r)$ ;
20.    $c = c \cup \{r\}$ ;
21. end while;
22. return result;
End;

```

```

Function find_best_record ( $S, c$ )
Input: a set of records  $S$  and a cluster  $c$ .
Output: a record  $r \in S$  such that  $IL(c \cup \{r\})$  is minimal.
1.  $n = |S|$ ;  $min = \infty$ ;  $best = null$ ;
2. for( $i = 1, \dots, n$ )
3.    $r = i$ -th record in  $S$ ;
4.   diff =  $IL(c \cup \{r\}) - IL(c)$ ;
5.   if( diff < min )
6.     min = diff;
7.     best =  $r$ ;
8.   end if;
9. end for;
10. return best;
End;

Function find_best_cluster ( $C, r$ )
Input: a set of clusters  $C$  and a record  $r$ .
Output: a cluster  $c \in C$  such that  $IL(c \cup \{r\})$  is minimal.
1.  $n = |C|$ ;  $min = \infty$ ;  $best = null$ ;
2. for( $i = 1, \dots, n$ )
3.    $c = i$ -th cluster in  $C$ ;
4.   diff =  $IL(c \cup \{r\}) - IL(c)$ ;
5.   if( diff < min )
6.     min = diff;
7.     best =  $c$ ;
8.   end if;
9. end for;
10. return best;
End;

```

Fig. 5. Greedy k -member clustering algorithm

Proof. Observe that the algorithm spends most of its time selecting records from the input set S one at a time until it reaches $|S| = k$ (Line 9). As the size of the input set decreases by one at every iteration, the total execution time T is estimated as:

$$T = (n - 1) + (n - 2) + \dots + k \approx \frac{n(n - 1)}{2}$$

Therefore, T is in $O(n^2)$. □

3.4 Improvement for Classification

In most k -anonymity work, the focus is heavily placed on the quasi-identifier, and therefore other attributes are often ignored. However, these attributes deserve more careful consideration. In fact, we want to minimize the distortion of quasi-identifier not only because the quasi-identifier itself is meaningful information, but also because a more accurate quasi-identifier will lead to good predictive models on the transformed table [6]. In fact, the correlation between the quasi-identifier and other attributes can be significantly weakened or perturbed due to the ambiguity introduced by the generalization of the quasi-identifier. Thus, it is critical that the generalization process does preserve the discrimination of classes using quasi-identifier. Considering this issue, Iyengar also proposed the *classification* metric (CM) as:

$$CM = \sum_{all\ rows} \text{Penalty}(row\ r) / N,$$

where N is the total number of records, and $\text{Penalty}(\text{row } r) = 1$ if r is suppressed or the class label of r is different from the class label of the majority in the equivalence group.

Inspired by this metric, we modify our algorithm in Figure 5 by replacing Line 4 of Function *find_best_record* with the following.

```
if (majority-class-label(c) == class-label(r))
    diff = IL({c ∪ {r}}) - IL(c);
else diff = IL({c ∪ {r}}) - IL(c) + classPenalty;
```

In essence, the algorithm is now forced to choose records with the same class label for a cluster, and the magnitude of enforcement is controlled by the weight of penalty. With this minor modification, our algorithm can effectively reduce the cost of classification metric without increasing much information loss. We show the results in Section 4.

4 Experimental Results

The main goal of the experiments was to investigate the performance of our approach in terms of data quality, efficiency, and scalability. To accurately evaluate our approach, we also compared our implementation with another algorithm, namely the *median partitioning algorithm* proposed in [8].

4.1 Experimental Setup

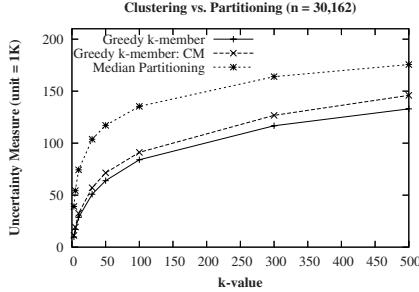
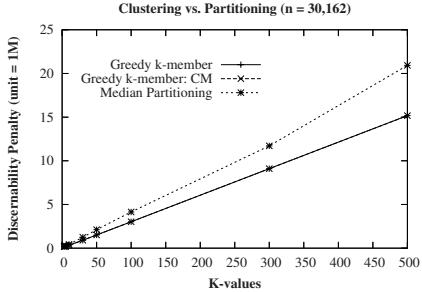
The experiments were performed on a 2.66 GHz Intel IV processor machine with 1 GB of RAM. The operating system on the machine was Microsoft Windows XP Professional Edition, and the implementation was built and run in Java 2 Platform, Standard Edition 5.0.

For our experiments, we used the Adult dataset from the UC Irvine Machine Learning Repository [10], which is considered a de facto benchmark for evaluating the performance of k -anonymity algorithms. Before the experiments, the Adult data set was prepared as described in [368]. We removed records with missing values and retained only nine of the original attributes. For k -anonymization, we considered $\{\text{age}, \text{work class}, \text{education}, \text{marital status}, \text{occupation}, \text{race}, \text{gender}, \text{and native country}\}$ as the quasi-identifier. Among these, *age* and *education* were treated as numeric attributes while the other six attributes were treated as categorical attributes. In addition to that, we also retained the *salary class* attribute to evaluate the classification metric.

4.2 Data Quality and Efficiency

In this section, we report experimental results on the greedy k -members algorithm for data quality and execution efficiency.

Fig. 6 reports the Total-IL costs of the three algorithms (median partitioning, greedy k -member, and greedy k -member modified to reduce classification error)

**Fig. 6.** Information Loss Metric**Fig. 7.** Discernibility Metric

for increasing values of k . As the figure illustrates, the greedy k -members algorithm results in the least cost of the Total-IL for all k values. Note also that the Total-IL cost of the modified greedy k -member is very close to the cost of the unmodified algorithm. The superiority of our algorithms over the median partitioning algorithm results from the fact that the median partitioning algorithm considers the proximity among the data points only with respect to a single dimension at each partitioning.

Another metric used to measure the data quality is the *Discernibility* metric (DM) [3], which measures the data quality based on the size of each equivalence class. Intuitively data quality diminishes as more records become indistinguishable with respect to each other, and DM effectively captures this effect of the k -anonymization process. Fig. 7 shows the DM costs of the three algorithms for increasing k values. As shown, the two greedy k -member algorithms perform better than the median partitioning algorithm. In fact, the greedy k -member algorithms always produce equivalence classes with sizes very close to the specified k , due to the way clusters are formed.

Fig. 8 shows the experimental result with respect to the CM metric described in Section 3. As expected, the greedy k -member algorithm modified to minimize classification errors (as described in Section 3) outperforms all the other algorithms. Observe that even without the modification, the greedy k -members algorithm still produces less classification errors than the median partitioning for every k value. We also measured the execution time of the algorithms for different k values. The results are shown in Fig. 9. Even though the execution time for the greedy k -member algorithm is higher than the partitioning algorithm, we believe that it is still acceptable in practice as k -anonymization is often considered an off-line procedure.

4.3 Scalability

Fig. 10 and 11 show the Total-IL costs and execution-time behaviors of the algorithms for various table cardinalities (for $k = 5$). For this experiment, we used the subsets of the Adult dataset with different sizes. As shown, the Total-IL costs increase almost linearly with the size of the dataset for both algorithms. However, the greedy k -member algorithm introduces the least Total-IL cost for

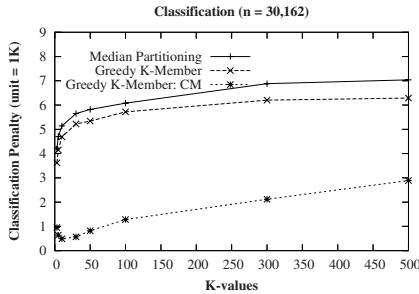


Fig. 8. Classification Metric

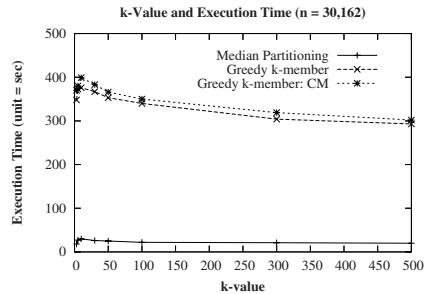


Fig. 9. Execution Time

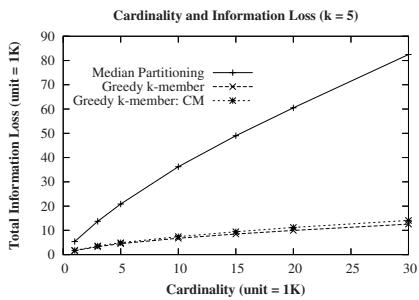


Fig. 10. Cardinality and Information Loss

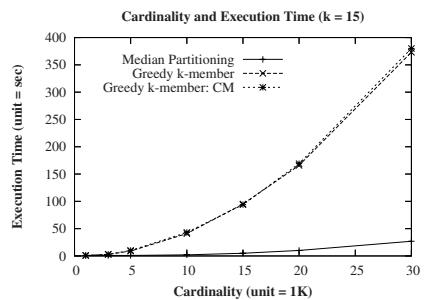


Fig. 11. Cardinality and Runtime

any size of dataset. Although the greedy k -members is slower than the partitioning algorithm, we believe that the overhead is still acceptable in most cases considering its better performance with respect to the Total-IL metric.

5 Conclusions

In this paper, we proposed an efficient k -anonymization algorithm by transforming the k -anonymity problem to the k -member clustering problem. We also proposed two important elements of clustering, that is, distance and cost functions, which are specifically tailored for the k -anonymization problem. We emphasize that our cost metric, IL metric, naturally captures the data distortion introduced by the generalization process and is general enough to be used as a data quality metric for any k -anonymized dataset.

References

1. G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Anonymizing tables. In *International Conference on Database Theory*, pages 246–258, 2005.
2. C. C. Aggrawal and P. S. Yu. A condensation approach to privacy preserving data mining. In *International Conference on Extending Database Technology*, 2004.

3. R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *International Conference on Data Engineering*, 2005.
4. B. C. M. Fung, K. Wang, and P. S. Yu. Top-down specialization for information and privacy preservation. In *International Conference on Data Engineering*, 2005.
5. Z. Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(2):283–304, 1998.
6. V. S. Iyengar. Transforming data to satisfy privacy constraints. In *ACM Conference on Knowledge Discovery and Data mining*, 2002.
7. K. LeFevre, D. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *ACM International Conference on Management of Data*, 2005.
8. K. LeFevre, D. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *International Conference on Data Engineering*, 2006.
9. A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *ACM Symposium on Principles of Database Systems*, 2004.
10. C. B. S. Hettich and C. Merz. UCI repository of machine learning databases, 1998.
11. P. Samarati. Protecting respondent's privacy in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13, 2001.
12. L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.
13. L. Sweeney. K-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.

Privacy Preserving Data Mining of Sequential Patterns for Network Traffic Data

Seung-Woo Kim¹, Sanghyun Park^{1,*}, Jung-Im Won², and Sang-Wook Kim²

¹ Department of Computer Science

Yonsei University, Korea

{kimsw,sanghyun}@cs.yonsei.ac.kr

² College of Information and Communications

Hanyang University, Korea

{jiwon,wook}@hanyang.ac.kr

Abstract. As a total amount of traffic data in networks has been growing at an alarming rate, many researches to mine traffic data with the purpose of getting useful information are currently being performed. However, since network traffic data contain the information about Internet usage patterns of users, network users' privacy can be compromised during the mining process. In this paper, we propose an efficient and practical method for privacy preserving sequential pattern mining on network traffic data. In order to discover frequent sequential patterns without violating privacy, our method uses the *N-repository server model* that operates as a single mining server and the *retention replacement technique* that changes the answer to a query probabilistically. In addition, our method accelerates the overall mining process by maintaining the *meta tables* in each site. Extensive experiments with real-world network traffic data revealed the correctness and the efficiency of the proposed method.

Keywords: Data mining, Sequential pattern, Network traffic, Privacy.

1 Introduction

Owing to the rapid advance of network technology, the number of computers connected to the Internet increases dramatically, so does the information delivered over the vast Internet. Recently, there has appeared a new kind of data mining researches that extract useful knowledge from network traffic data automatically gathered by a remote server [9,7,11].

Table 1 shows an example of network traffic data gathered by Ethereal¹. The network traffic data have the following characteristics: First, there exist various kinds of data since all the computers connected to the Internet can produce network traffic data potentially. Second, a huge amount of network traffic data

* Corresponding Author.

¹ <http://www.ethereal.com/>

Table 1. An example of network traffic data gathered by Ethereal

timestamp	source address	source port	destination address	destination port
13:37:11.950966	180.1.1.1	36872	amazon.com	www
13:37:11.954474	amazon.com	www	180.1.1.1	36872
13:37:22.384472	180.1.1.1	36915	192.168.1.3	telnet
13:37:22.385327	192.168.1.3	telnet	180.1.1.1	36915

are accumulated due to frequent actions for data sending/receiving by a lot of computers. Third, network traffic data are scattered over a large number of sites.

Sequential pattern mining is the most useful for this application since the order of events has an important meaning in network traffic data [9,7].

Network traffic data contain detailed information of Internet usage for every user, which informs that a user accesses a site at a time specifically. Herein, data mining on network traffic data has the problem of compromising privacy of network users. Therefore, it requires sophisticated techniques for hiding or reforming users' private information during a data gathering process. Moreover, these techniques should not sacrifice the correctness of mining results.

Privacy preserving data mining is a new kind of a research area that aims at mining data with guaranteeing privacy of individual users [4,13,25,8,6,10,12,14]. Recently, there have been many research efforts performed in this area. Most methods proposed in prior studies, however, manage data in a few sites or deal with a small number of distinct types of data. Thus, these methods are not appropriate for mining network traffic data since they suffer from the problems of incorrectness and low performance.

In this paper, we discuss solutions to the problems that occur in previous methods. We propose a novel method for sequential pattern mining on network traffic data. The proposed method preserves privacy of sites and guarantees the correctness of mining results. The method discovers frequently-occurring network traffic patterns with hiding site information through two ways: (1) It employs the *N*-repository server model that makes multiple servers behave as a single mining server; (2) It uses the retention replacement technique that changes the answer by a given probability. Also, the method maintains *meta tables* in each site so as to quickly determine whether candidate patterns ever occurred in the site, thereby making the overall mining process become highly efficient.

2 Related Work

Clifton et al. [4] firstly raised the privacy problem in data mining and motivated subsequent studies [13,25,8,6,10,12,14] that aimed to solve the problem.

In the method proposed in [2], in order to preserve privacy, each site changes the original numeric value of an individual item before sending the value to the server by adding an arbitrary value selected from given probability distribution. The server builds a decision tree by reconstructing actual value distribution.

Other method called the *retention replacement* [13][3] perturbs and reconstructs data in gathering and mining, respectively, for privacy preservation. For every data whose element represents 0 or 1, each site sends the original value with probability p and the perturbed one with probability $(1 - p)$. For gathered data, the server counts the total numbers of 1's and 0's, and then estimates the original numbers of 1's and 0's. This method is applicable only to boolean data.

Some later studies [5][3] tried to apply those two methods to various data types, however, showed lower accuracy as the number of data types increases.

The method proposed by Rizvi et al. [13] uses the *retention replacement* for finding frequent itemsets. This can be applied to the case where item types to occur are pre-determined. Considering network traffic data where a large number of item types occur, we can hardly determine all the item types in advance. Also, this method finds frequent patterns via a whole database scan and thus is very inefficient since network traffic data are very huge.

The method proposed in [8] collects local frequent itemsets from sites by employing a commutative encryption and obtains global frequencies of itemsets by employing a secure sum which uses a random number. For performing a commutative encryption and a secure sum, this method has to serially send data in the cycle of sites. This requires a lot of time in case of a large number of sites. Fukasawa et al. [6] improved the efficiency and security of this method. However the improved one still has cycling communications.

Zhan et al. proposed a method for sequential pattern mining with privacy preservation [14]. This method mainly targets a distributed environment where vertical partitioning without duplication is employed. In our situation, duplicated data could occur in more than one site since multiple PCs can access the same Internet site. Therefore, this method is inapplicable to network traffic data.

In the method proposed in [10], a secure protocol is used for mining a decision tree classifier from distributed sites. Pinkas [12] showed how protocols for secure distributed computation can be employed for privacy preservation, however he also pointed out that the performance of protocols should be improved.

In summary, prior studies have the problems applying to a large amount of network traffic data. First, due to a variety of data types, previous methods are not directly applicable and cannot obtain accurate mining results. Second, since there exist a large number of sites and data can be duplicated, previous methods targeted for a distributed database environment have limitations on practicality.

3 Problem Definition

Network traffic data are normally gathered by a tcp/ip data capture program such as Ethereal. In this paper, we aim at finding sequential patterns from network traffic data without disclosing data in each site. First, we simplify the network traffic data in the form of Table 1 as those in the form of Table 2. In Table 2, “out” denotes sending and “in” does receiving.

In order to find temporal relationship among events in network traffic data, we can apply sequential pattern mining methods [9][7]. We impose a restriction that

Table 2. An example of network traffic data reconstructed

timestamp	in/out	address	timestamp	in/out	address
13:37:11.950966	out	amazon.com	13:37:22.384472	out	192.168.1.3
13:37:11.954474	in	amazon.com	13:37:22.385327	in	192.168.1.3

two adjacent items in network traffic data should have a time interval smaller than or equal to a predefined *MaxGap* value to be regarded as related.

We formulate the problem we are going to solve as follows: Given t sites T_1, T_2, \dots, T_t , the maximum time interval *MaxGap*, and the minimum support *MinSup*, we discover all the sequential patterns, which have a support larger than *MinSup* and a time interval between any pair of adjacent items equal to or smaller than *MaxGap*. We assume that a site stores network traffic data as in the form of Table 2.

A mining process should also satisfy the condition for preserving privacy in every site. Let us denote a set of sites, where network traffic has occurred, as E and a set of network traffic data as I . In a mining process, an element e_j in E is opened since it participates in the mining process; Also, an element i_k in I is also opened since it should be contained in a result of mining. However, a pair of (e_j, i_k) , which says a site e_j has been connected to an IP address i_k , should not be opened in a mining process. We define this condition for preserving privacy.

4 Proposed Method

4.1 Overall Mining Process

The proposed mining process consists of four phases. Table 3 shows the definition of symbols which are used to explain the mining process. The first phase utilizes the *N-repository server model* to safely discover F_1 . The second phase generates C_{k+1} by self-joining F_k . k is initialized to 1 when the second phase is executed for the first time. If C_{k+1} is empty, we enter into the final phase. Otherwise, we enter into the third phase. For each candidate in C_{k+1} , the third phase sends every site the query asking whether the candidate has ever occurred in the site. After receiving the answers from all sites, the third phase judges whether each candidate is frequent or not, and then constructs F_{k+1} , with the candidates

Table 3. Definition of symbols

F_1	A set of all frequent sequential patterns of length 1 (or large 1-sequences, frequent items)
F_k	A set of all frequent sequential patterns of length k (or large k -sequences)
C_k	A set of all candidate patterns of length k

judged as frequent. The third phase then increases k by 1 and calls the second phase. The final phase prints all frequent patterns in F_1, F_2, \dots , and F_k , and stops the mining process.

4.2 Finding Frequent Items Using *N-Repository Server Model*

The proposed *N-repository server model* finds F_1 , without compromising the condition for preserving privacy by concealing the linkage between the site identifier and the traffic data, (e_j, i_k) . More specifically, it obscures their linkage by encrypting the traffic data, i_k , at the first step and by aggregating the site identifiers, e_j , at the second step.

The proposed *N-repository server model* consists of N servers, $\{S_1, S_2, \dots, S_N\}$, and N pairs of encryption keys and decryption keys, $\{(EK_1, DK_1), (EK_2, DK_2), \dots, (EK_N, DK_N)\}$. Each site has all encryption keys but server S_i has only a decryption key DK_i ($1 \leq i \leq N$). To find frequent items safely, the *N-repository server model* operates as follows:

1. Each site classifies the items (i.e., the traffic data) into N groups, $\{G_1, G_2, \dots, G_N\}$, using a hash function.
2. Each site encrypts the items in G_i with encryption key EK_i ($1 \leq i \leq N$).
3. Each site sends the encrypted items in G_i to server S_{i+1} ($1 \leq i \leq N-1$) and the encrypted items in G_N to server S_1 .
4. Each server performs the aggregation on the encrypted items to obtain the number of occurrences of each encrypted item and then picks up the encrypted *frequent* items.
5. Each server S_i sends encrypted *frequent* items to server S_{i-1} ($2 \leq i \leq N$) and server S_1 does to server S_N .
6. Each server S_i decrypts the received items with its decryption key DK_i and then reports the frequent items to public.

We assume that the servers in our model operate in a *semi-trusted* operation model. In the *semi-trusted* operation model, servers may try to acquire private data but do not cooperate with other servers to do that. This semi-trusted operation model is common in real environments where one wants to get the result of computation but is not willing to offer one's own data to others [14].

4.3 Finding Frequent Patterns Longer Than One

After finding out F_1 , we have to sequentially discover frequent patterns longer than one. At first, one of N servers is elected as a principal mining server. To discover all frequent patterns longer than one, the principal mining server assigns 1 to variable k and executes the following steps.

1. It produces C_{k+1} by self-joining F_k in the same way as *Apriori* algorithm [1]. It executes step 5 if C_{k+1} is empty. Otherwise, it executes step 2.
2. For each candidate pattern CP in C_{k+1} , the server sends every site T a query asking whether CP has ever occurred in T or not.

3. Each site T sequentially inspects traffic data or the *meta tables*, which will be described in Section 4.4, to determine CP 's occurrence in T . An actual answer of the query would be 1 or 0 as CP 's occurrence. However, to preserve the privacy of the site, the actual answer is perturbed by the *retention replacement* [133].
4. For each query, the principal mining server aggregates the counts of the sites answered 1 and the sites answered 0. Then, using the two counts, it conjectures the number of sites whose actual answers were 1. It then compares that number with $MinSup$ and constructs F_{k+1} , by choosing from C_{k+1} . It finally increases k by 1 and calls step 1.
5. When it reaches this step, C_{k+1} is empty. Therefore, it prints all the frequent patterns discovered and then stops the execution of the algorithm.

4.4 Meta Tables to Quickly Determine the Occurrence or Non-occurrence of Candidate Patterns

In the *Apriori* algorithm, patterns of length k can be regarded as candidate patterns only when all of their sub-patterns are frequent. However, in the sequential pattern mining with time constraints, even the patterns containing infrequent sub-patterns can be treated as candidate patterns if all of their sub-patterns occurring *contiguously* in the underlying patterns are frequent. Therefore, compared to the mining techniques based on the original *Apriori* algorithm, the sequential pattern mining with time constraints impose less requirement for patterns to be treated as candidate patterns. As a result, more candidate patterns are generated and, to accelerate the overall mining process, it is crucial to handle each candidate pattern efficiently.

In this paper, we employ special-purpose *meta tables* in each site T for speeding-up the process to decide the occurrence or non-occurrence of CP in T .

Meta tables for storing pairs of items satisfying *MaxGap*

Let m denote the number of frequent items. At first, the principal mining server sends out the list of all frequent items to each site T . Then, site T lexicographically sorts the frequent items and assigns each frequent item the corresponding lexicographic order. Site T then stores the name and lexicographic order of each frequent item into the meta table called **FreqItems**. **FreqItems** consists of two columns, **ItemName** and **Order**. Given a frequent item, **ItemName** and **Order** store its name and lexicographic order, respectively.

The second meta table is **OccTs_OccBits**. This table consists of three columns, **Order**, **OccTs**, and **OccBits**. For each frequent item FI in the traffic data of T , **Order** stores the lexicographic order of FI , and **OccTs** stores the timestamp at which FI occurred, and **OccBits** stores a bit-vector of length m whose i^{th} bit indicates whether or not the frequent item of lexicographic order i has ever occurred within *MaxGap* after the occurrence of FI . We denote the i^{th} bit of **OccBits** as $OccBits(i)$. **OccTs_OccBits** can be constructed by scanning the entire traffic data in site T . The number of tuples in **OccTs_OccBits** is same as the number of occurrences of frequent items in T .

The third meta table is OccCnts . OccCnts consists of $m + 1$ columns, Order, Cnt_1 , Cnt_2 , ..., Cnt_m . OccCnts has a tuple for each frequent item and therefore contains m tuples. Let us consider the i^{th} tuple of OccCnts . It has i as a value of Order. As a value of Cnt_j , it has the number of occurrences of the frequent item of order j whose timestamps are within MaxGap after the occurrences of the frequent item of order i . The i^{th} tuple of OccCnts can be populated by querying OccTs_OccBits .

An example of *meta tables* maintained within a single site is shown in Fig. □

< FreqItems >		< OccTs_OccBits >				< OccCnts >					
ItemName	Order	Order	OccTs	OccBits	Order	OccTs	OccBits	Order	Cnt ₁	Cnt ₂	Cnt ₃
A	1	1	13:37:11.95	000	2	13:38:17.23	010	1	0	2	1
B	2	1	13:37:32.43	010	2	13:38:19.12	000	2	1	1	1
C	3	1	13:38:07.05	001	3	13:37:35.17	000	3	1	0	1
		1	13:38:15.51	010	3	13:38:08.54	000				
		2	13:37:34.21	001	3	13:38:12.31	001				
		2	13:38:05.44	100	3	13:38:14.08	100				

Fig. 1. An example of *meta tables* maintained within a single site

Determining the occurrences or non-occurrences of candidate patterns

The algorithm to determine the occurrences or non-occurrences of candidate patterns using the *meta tables* has the following 4 steps.

1. Divide a candidate pattern

Let $CP_n = \langle I_1, I_2, \dots, I_n \rangle$ denote a candidate pattern with n items. We first divide CP_n into $n - 1$ sub-patterns each of which consists of two adjacent items of CP_n . The j^{th} sub-pattern of CP_n is denoted as $SP_j = \langle I_j, I_{j+1} \rangle$ ($j = 1, 2, \dots, n - 1$).

2. Determine the execution orders of the sub-patterns

Sub-patterns are executed on the *meta tables* and their results are joined with those of other sub-patterns. If we are able to discover the optimal execution orders which minimize the sizes of intermediate results, then we can determine the occurrence or non-occurrence of a candidate pattern as early as possible. The simplest way to find out the optimal execution orders is to consider all the possible combinations of execution orders and to choose the one which will produce the smallest intermediate results. However, there are $(n - 1)!$ distinct combinations for $n - 1$ sub-patterns and thus such a simple approach is not scalable to large n . Therefore, we employ the following *greedy* algorithm which quickly discovers near-optimal execution orders.

- (a) We choose the sub-pattern which will have the smallest result set size, and let 1 be its execution order. We then assign 1 to variable k .
- (b) Let us assume that the execution order of SP_j has just been decided as k . To decide the sub-pattern of execution order $k + 1$, we decrease j'

from $j - 1$ to 1 until we find $SP_{j'}$ whose execution order is not decided yet. Also, we increase j'' from $j + 1$ to $n - 1$ until we find $SP_{j''}$ whose execution order is not decided yet.

- (c) If neither $SP_{j'}$ nor $SP_{j''}$ exists, we stop the execution of the *greedy* algorithm. However, if $SP_{j''}$ does not exist but $SP_{j'}$ exists, then we let $k + 1$ be the execution order of $SP_{j'}$. On the contrary, if $SP_{j'}$ does not exist but $SP_{j''}$ exists, then we let $k + 1$ be the execution order of $SP_{j''}$. If both $SP_{j'}$ and $SP_{j''}$ exist, then we choose the one which will have a smaller result set size and let $k + 1$ be its execution order. If their result set sizes will be same, then we choose the one farther from the corresponding end. This reduces the possibility of the absence of either $SP_{j'}$ or $SP_{j''}$ in the next step and therefore enables to obtain a better combination of execution orders.
- (d) We increase k by one and return to step 2(b).

In the middle of this *greedy* algorithm, there is a step to calculate the result set sizes of sub-patterns. The result set size of $SP_j = \langle I_j, I_{j+1} \rangle$ is equal to the number of occurrences of item I_{j+1} within *MaxGap* after the occurrences of item I_j . The result set size of a sub-pattern can be easily obtained by using two *meta tables*, *FreqItems* and *OccCnts*, who were explained above.

3. Execute the sub-patterns and join their results

According to the execution orders obtained in step 2, we execute all sub-patterns one by one while joining their intermediate results. That is, for each k from 1 to $n - 1$, we execute the following steps.

- (a) For the two items of the sub-pattern whose execution order is k , we find their lexicographic orders using *FreqItems*. Let p and q be the lexicographic orders of the preceding item and the succeeding item, respectively.
- (b) We execute the following SQL statement to obtain the result set RS_k of the sub-pattern of execution order k .

```
select p, OccTs, q      // p and q are not column names but constants
into RSk
from OccTs_OccBits
where Order = p and OccBits(q) = 1;
```

- (c) We join the result set RS_k with JRS_{k-1} , the intermediate result set obtained by sequentially joining all the result sets of sub-patterns of execution orders from 1 to $k - 1$, producing a new intermediate result set JRS_k . For a simpler explanation, let us rename the tables to be joined as follows. If the sub-pattern of execution order k is on the left of the sub-patterns of execution orders from 1 to $k - 1$, then we rename the sub-pattern of execution order k as **TA** and the intermediate result set JRS_{k-1} as **TB**. Otherwise, we rename the sub-pattern of execution order k as **TB** and the intermediate result set JRS_{k-1} as **TA**. Then, the conditions for a tuple **ta** of **TA** to be joined with a tuple **tb** of **TB** are like the following:

- **Join condition 1:** The last item of ta must be identical to the first item of tb .
 - **Join condition 2:** The gap from the timestamp of tb 's first item to the timestamp of ta 's last item must not be larger than $MaxGap$.
- (d) We check if the join result JRS_k is empty. If so, we proceed to step 4. Otherwise, we increase k by one and return to step 3(a).
4. Determine the occurrence or non-occurrence of a candidate pattern

We check if the final result set of step 3 is empty. If so, we conclude that the candidate pattern being considered has never occurred in this site. Otherwise, we judge that there is at least one occurrence of the candidate pattern.

Meta table to quickly judge the non-occurrence of a candidate pattern

Apriori algorithm joins frequent patterns of length n with themselves to generate candidate patterns of length $n + 1$.

Let $\{CP_n\}$ denote the set of candidate patterns of length n . Also, let $\{CP'_n\}$ denote the set of candidate patterns in $\{CP_n\}$ whose occurrences were detected in the site. Now, let us consider a candidate pattern of length $n + 1$, CP_{n+1} , delivered to the site most recently. If we break CP_{n+1} into two sub-patterns of length n , $CP_{n+1}[1..n]$ and $CP_{n+1}[2..n+1]$, then both of them are certainly elements of $\{CP_n\}$. The prerequisites for CP_{n+1} to occur in the site are the occurrences of both $CP_{n+1}[1..n]$ and $CP_{n+1}[2..n+1]$. Therefore, if either $CP_{n+1}[1..n] \notin \{CP'_n\}$ or $CP_{n+1}[2..n+1] \notin \{CP'_n\}$ is satisfied, then we can quickly recognize the non-occurrence of CP_{n+1} without looking up the *meta tables*. We implement this pruning method by maintaining a meta table named **OccCandPatt** in each site. **OccCandPatt** stores the string representation of each candidate pattern whose occurrence has ever detected in the underlying site.

This pruning method enables to quickly judge the non-occurrence of a candidate pattern but increases the size of **OccCandPatt** continually. However, note that this method requires only the candidate patterns of length n in order to determine the non-occurrence of a candidate pattern of length $n + 1$. Therefore, when the site receives from the principal mining server a candidate pattern of length $n + 1$ for the first time, it removes the candidate patterns of length $n - 1$.

5 Performance Evaluation

5.1 Environment for Experiments

In experiments, we collected 5,024,295 traffic data by Ethereal during 5 days. From them, we extracted 747,000 traffic data related to 736 IP addresses. The average inter-arrival time is 462.38 msec.

We compared the performances of three methods: **Naive**, **OccTs**, and **OccTs+OccCandPatt**. In order to discover F_1 , **Naive** uses the *retention replacement* for all traffic data. Furthermore, it scans the original traffic data to verify whether every candidate is actually frequent. **OccTs** discovers F_1 by using the *N-repository server model* and F_k ($k \geq 2$) by the *retention replacement*. **OccTs**

decides whether a candidate pattern is frequent by searching *meta tables* OccTs_OccBits and OccCnts. Finally, OccTs+OccCandPatt, which is basically based on OccTs, additionally uses meta table OccCandPatt. Furthermore, both OccTs and OccTs+OccCandPatt employ a *greedy* algorithm to determine the execution order of sub-patterns.

As a measure for evaluating accuracy, we used *Recall* and *Precision*. *Recall* indicates how much fraction are mined from all the actually frequent ones. *Precision* indicates how much fraction of mined patterns are actually frequent. As a performance measure, we used the average elapsed times in mining sequential patterns of the maximum length 6.

The hardware platform is the Pentium IV 3.0GHz PC equipped with 512 Mbytes main memory and 80 Gbytes hard disk of 7200RPM. The software platform is the Windows XP and the Java 2 Runtime Environment 1.4.2.

Since we got quite good performances in our parameter setting experiments, we set *MinSup*, *MaxGap*, the number of sites, and the number of servers to 0.2, 20, 10, and 5, respectively in the following experiments.

5.2 Analysis of Accuracy

In order to evaluate accuracy of the proposed *N-repository server model*, we compared *Recall* and *Precision* of OccTs and Naive. Because the accuracy of both OccTs and OccTs+OccCandPatt is the same, we show only Naive and OccTs.

We evaluated *Recall* and *Precision* with different probability p . We set the number of sites to 50. Fig. 2 shows the results with p of 0.51 to 1. We note that the *retention replacement* is inapplicable with p of 0.5 [13].

The results show that, in Naive and OccTs, both *Recall* and *Precision* get higher as p gets close to 1. This is due to the *retention replacement* used in both methods to find frequent sequential patterns whose length is longer than 1. OccTs performs 1.04 to 1.20 and 1.01 to 1.12 times better than Naive in *Recall* and *Precision*, respectively.

Naive is inapplicable to analyzing real Internet traffic data because it has to know all the items likely to occur in advance. Furthermore, in the above two experiments, OccTs showed accuracy higher than Naive.

5.3 Analysis of Performance

In order to evaluate the performance of OccTs and OccTs+OccCandPatt, we compared them with Naive in terms of the elapsed time for mining. We measured the elapsed time with different numbers of traffic data in each site. We set probability p in *retention replacement* to 1 in order to evaluate the average elapsed times of all the methods fairly. Fig. 3 shows the result.

We observe that, in all three methods, as the volume of traffic data gets larger, the elapsed time increases. This is because more frequent patterns appear with a larger volume of traffic data. OccTs performed 1.60 to 2.38 times better than Naive. It stores all pairs of frequent items that occur within *MaxGap* into a

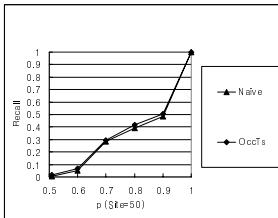


Fig. 2. Recall and Precision with different probability p

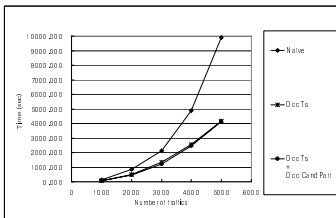
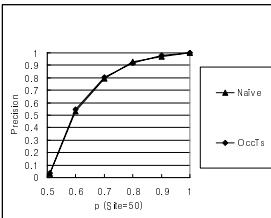


Fig. 3. The elapsed time with different numbers of traffic data

meta table *OccTs_OccBits* and quickly determines whether candidate patterns occur by joining these pairs without accessing the network traffic data.

OccTs+OccCandPatt ran 1.01 to 1.10 times faster than *OccTs*. By referring to *OccCandPatt*, it examines whether candidate patterns have ever occurred in the site before searching *OccTs_OccBits*. Therefore, it achieves the pruning effect in the mining process. That is, the total elapsed time decreases because the number of candidate patterns to be searched in *OccTs_OccBits* gets smaller.

6 Concluding Remarks

In this paper, we have proposed a practical method for sequential pattern mining on network traffic data. The proposed method preserves privacy of sites and provides high accuracy of mining results. The proposed method can be used for finding frequent sequential visiting patterns of web pages. The mining results can be applied to prefetching of web pages and load balancing in web servers.

The contributions of the paper are summarized as follows: First, we have proposed a privacy preserving method that mines frequent sequential patterns from network traffic data. Our method uses the *N-repository server model* that operates as a single mining server and also employs the *retention replacement technique* that changes the answer by a given probability. Second, we have designed *meta tables* maintained in each site so as to quickly determine whether candidate patterns ever occurred in the site. Third, we have demonstrated the correctness and the efficiency of the proposed method via extensive experimentation with real-world network traffic data.

Acknowledgements

This work was supported by the Seoul R&BD Program(10561) in 2006, Korea Research Foundation Grant funded by Korea Government (MOEHRD, Basic Research Promotion Fund) (KRF-2005-206-D00015), and the MIC of Korea under the ITRC support program supervised by the IITA (IITA-2005-C1090-0502-0009).

References

1. R. Agrawal and R. Srikant, "Mining Sequential Patterns," In Proceedings of the 11th IEEE International Conference on Data Engineering, pp. 3–14, 1995.
2. R. Agrawal and R. Srikant, "Privacy-Preserving Data Mining," In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 439–450, 2000.
3. R. Agrawal, R. Srikant, and D. Thomas, "Privacy Preserving OLAP," In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 251–262, 2005.
4. C. Clifton and D. Marks, "Security and Privacy Implication of Data Mining," In Proceedings of the 1996 ACM Workshop on Data Mining and Knowledge Discovery, pp. 15–19, 1996.
5. A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke, "Privacy Preserving Mining of Association Rules," In Proceedings of the 2002 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 217–228, 2002.
6. T. Fukasawa, J. Wang, T. Takata, and M. Miyazaki, "An Effective Distributed Privacy-Preserving Data Mining Algorithm," In Proceedings of the 5th International Conference on Intelligent Data Engineering and Automated Learning, pp. 320–325, 2004.
7. Y. Hu and B. Panda, "A Data Mining Approach for Database Intrusion Detection," In Proceedings of the 2004 ACM Symposium on Applied Computing, pp. 711–716, 2004.
8. M. Kantarcioglu and C. Clifton, "Privacy-Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data," In Proceedings of the 2002 ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, pp. 24–31, 2002.
9. W. Lee, S. Stolfo, and K. Mok, "A Data Mining Framework for Building Intrusion Detection Models," In Proceedings of IEEE Symposium on Security and Privacy, pp. 120–132, 1999.
10. Y. Lindell and B. Pinkas, "Privacy Preserving Data Mining," In Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology, pp. 36–54, 2000.
11. J. Luo and S. Bridges, "Mining Fuzzy Association Rules and Fuzzy Frequency Episodes for Intrusion Detection," International Journal of Intelligent Systems, Vol. 15, No. 8, pp. 687–704, 2000.
12. B. Pinkas, "Cryptographic techniques for privacy-preserving data mining," ACM SIGKDD Explorations Newsletter, Vol. 4, No. 2, pp. 12–15, 2002.
13. S. Rizvi and J. Haritsa, "Maintaining Data Privacy in Association Rule Mining," In Proceedings of the 28th International Conference on Very Large Data Bases, pp. 682–693, 2002.
14. J. Zhan, L. Chang, and S. Matwin, "Privacy-Preserving Collaborative Sequential Pattern Mining," In Proceedings of SIAM International Workshop on Link Analysis, Counter-terrorism, and Privacy, pp. 61–72, 2004.

Privacy Preserving Clustering for Multi-party

Weijia Yang¹ and Shangteng Huang²

¹ Shanghai Jiao Tong University, Shanghai 200030, China

weijia.yang@yahoo.com.cn

² Shanghai Jiao Tong University, Shanghai 200030, China

huang-st@cs.sjtu.edu.cn

Abstract. Privacy concerns on sensitive data are now becoming indispensable in data mining and knowledge discovering. Data owners usually have different concerns for different data attributes. Meanwhile the collusion among malicious adversaries produces a severe threat to the security of data.

In this paper, we present an efficient method to generate the attribute-wised orthogonal matrix for data transformation. Moreover, we introduce a privacy preserving method for clustering problem in multi-party condition. Our method can not only protect data in the semi-honest model but also in the malicious one. We also analyze the accuracy of the results, the privacy levels obtained, and their relations with the parameters in our method.

1 Introduction

Data mining is a powerful tool in discovering hidden patterns from large amount of data. It can also be a threat when not used properly. Therefore, privacy preserving data mining is becoming a popular research direction these years. Starting from researches [12], the statistical and cryptographic theories have been extensively applied. Consequently, quite a few data protecting methods have been presented. Different methods correspond to different scenarios: secure multi-party calculation (SMC) for distributed data mining; perturbation techniques for data publishing.

In this paper, we mainly focus on such scenario: several parties send their data records to the mining server in order to get an overall aggregate model, while each party has its own privacy concerns about its data. Before transmitting, they will disguise their data by transformation matrix. And the miner has to build the right model based on these perturbed data. The mining task we consider in this paper is clustering.

We consider the owners have different privacy requirements for each data attributes, and we implement the variable-wised privacy based on the orthogonality of the transforming matrix. We also analyze the impacts on privacy from both semi-honest and malicious participants in our privacy preserving clustering method. The randomization techniques are also applied to rebuild the privacy level in face of complicity.

This paper is organized as follows. In Section 2, we provide the works related to our topics. The attribute-wised data transformation algorithm is presented in Section 3. In addition, the privacy measurement is also given. In Section 4, we introduce in detail our new privacy preserving method for clustering. Section 5 analyzes the accuracy impact our method brings, and the privacy levels obtained both in semi-honest and malicious conditions are discussed in Section 6. Section 7 presents our experimental results. Finally, we summarize the conclusions of our study in Section 8.

2 Related Works

Many works have been made on the problem of privacy preserving for different kinds of data mining tasks. While in this paper, our scenario is similar with the works [3][4][5][6] which are based on Randomized Response techniques [7]. In the work of [3][6], the data owner transmits the real boolean data in a fixed probability θ , so that the others can not directly get the original data while the server can rebuild the statistics knowing θ . And different transmitting probabilities were placed on different data attributes in [5].

The work in [8] proposed a rotation based way like spatial transformation. This method transforms the data into another coordinates system, so that it keeps the similarity between records and protects the original. The work in [9] applied the orthogonal transformation way to classification problem, it proved several classifiers to be rotation-invariant, and presented an algorithm to reach higher privacy levels. The work in [10] generated its transformation matrix by independent and identically distributed (i.i.d.) variables(i.e. each variable has the same probability distribution as the others and all are mutually independent) and projected the original data from high dimensions to lower ones without losing much accuracy.

Private Clustering has also been studied by cryptographic, such as using SMC in the work of [11].

Our work differs from the above papers in that: we put attribute-wised privacy concerns into random transformation matrix generation; the threats to privacy from both semi-honest and malicious parties are considered; we reinforce the orthogonal transformation method with randomization techniques.

The privacy concerns have also been studied in other mining tasks: [12][13] for association rule mining; [14] for Bayesian network; [1][15] for classification tree building; ... However, these are out of the scope of our topic.

3 Attribute-Wised Transformation Matrix

In this paper we'll mainly focus on one of the most common mining task: clustering. All parties are assumed to have homogeneous datasets. Considering different data attributes may have different privacy concerns, the participants should

carry out the transforming process with their own transformation matrixes which represent their privacy requirements respectively. Thus, there come two main questions:

1. How to generate the transformation matrix representing the attribute-wised privacy concerns?
2. Since subsets of data are distributed in different coordinate systems after the transformation, how does the miner successfully clustering the records into right groups?

3.1 Privacy Measurement

For a participant party, suppose $l \times m$ matrix X represents the original dataset with l rows and m columns, $l \times m$ matrix Y is the transformed dataset, X' and Y' are the unified version, $H(m \times m)$ is the transformation matrix. As defined in [9], we use the variance of the difference between X' and Y' to determine the privacy level of data variables.

$$\text{Privacy Level} \sim Cov(X' - Y') = Cov(X' \cdot (E - H)) \quad (E \text{ is the entity matrix}) \quad (1)$$

In practice, each attribute in a data set usually involves different weight of privacy which is hard to be quantified by the owner. But through his experience, the owner still can appropriately establish an order of priority. For example, a commercial bank has its records of customers consisting of age, income, deposit, credit ... It is probable for its officer to evaluate the risk of leakage of these attributes and sort them by privacy concerns in descending order like: deposit > credit > income > age > ... However, it's still quite hard to quantify the specific difference in between. E.g. whether the privacy level of deposit is exactly twice or 2.5 times higher than credit? Therefore, we take the order of privacy priority as the directive to generate the attribute-wised transformation matrix.

3.2 Attribute-Wised Transformation

The orthogonal transformation maintains the result of the dot product operation, which is helpful to clustering, and is also employed by [9]. Since (1) not only includes the column vectors of H , but also shows their positions (the order of the column vectors in E should change with that of the column vectors in H), we can't simply reorder the columns of an orthogonal matrix to achieve the attribute-wised privacy. In order to get an appropriate H for a dataset of m columns, we list all the conditions H should satisfy to form the equation set:

$$\left\{ \begin{array}{l} H_i^T \cdot H_i = 1 \quad (H_i, E_i \text{ represent the } i\text{th column vector of } H, E) \\ H_i^T \cdot H_j = 0 \quad (i \neq j) \\ Var(X \cdot (E_1 - H_1)) > Var(X \cdot (E_2 - H_2)) \\ \quad > \dots > Var(X \cdot (E_m - H_m)) \end{array} \right. \quad (2)$$

It takes high time complexity to locate one of the possible solutions of (2). We present a convenient way to achieve this. Firstly, we will generate a random

orthogonal matrix. The random orthogonal matrix generation can be found in the work of [16].

Lemma 1. Let D be a $l \times m$ unified data matrix(i.e. a dataset with l rows and m columns), E_i be i th column vector of identity matrix E , H_i be i th column vector of an $m \times m$ orthogonal transformation matrix H . Comparing with other attributes, the element $h_{i,i}$ in the transformation matrix determines the relative order of privacy for i th attribute in D .

Proof. The privacy of i th attribute is quantified as:

$$\text{Var}(D \cdot (E_i - H_i)) = (E_i - H_i)^T \cdot \text{Cov}(D) \cdot (E_i - H_i) . \quad (3)$$

For the purpose of compare, we can omit the $\text{Cov}(D)$ in (3). So we have:

$$\begin{aligned} & (E_i - H_i)^T \cdot (E_i - H_i) \\ &= E_i^T \cdot E_i + H_i^T \cdot H_i - E_i^T \cdot H_i - H_i^T \cdot E_i \\ &= 2 - E_i^T \cdot H_i - H_i^T \cdot E_i \\ &= 2 - 2 \cdot h_{i,i} . \end{aligned} \quad (4)$$

Thus, $-h_{i,i}$ represents the order of the privacy of each variable. \square

Moreover, the orthogonal matrix has a property that after changing the order of its column or row vectors, it still keeps its orthogonality. Therefore, by moving the columns and rows in the random orthogonal matrix, we sort the crucial elements to achieve the attribute-wised privacy. We also assume the attribute privacy levels are required to be arranged in descending order.

After getting a random orthogonal matrix H [16], our attribute-wised privacy transformation matrix generation algorithm executes as follows:

Input: data matrix D , random orthogonal matrix H

Output: attribute-wised orthogonal matrix G

1. Randomly choose or manually specify an element for every column in H so that there's only one element selected in each row, the selected elements make up vector S .
2. Sort the row vectors of H in ascending order according to their previously selected s_i (s_i represents the i th element in vector S).
3. Sort H 's column vectors in ascending order according to their selected s_i to form G .

Suppose the party has m attributes,then the time complexity of the algorithm is $O(m)$. In this way, we sort the diagonal elements in ascending order without losing the orthogonal feature of the matrix. Thus, the privacy levels of the variables are arranged in descending order. And a different order of privacy arrangement can also be achieved in the similar way.

4 Multi-party Clustering Protocol

In our situation, like questionnaire survey, the participants may not know each other; they just fill in papers and leave, not to be online in the same time. Thus, it's nearly impossible to run protocols or calculations among all of them. However, during the survey, one interviewee is much likely to meet the next one. Therefore, it makes sense to have the adjacent respondents share parts of their information. Our method is also enlightened from this idea.

In our method, every party has its data matrix right multiplied by an orthogonal matrix in order to retain the distance between its record vectors.

Considering the two-party condition, suppose party A has data matrix X_A , party B has X_B , H_A, H_B are the corresponding orthogonal matrixes. Then, $Y_A = X_A \cdot H_A, Y_B = X_B \cdot H_B$.

Before transformation, row vectors in X_A and X_B can be viewed as points in the vector space with bases $\{(1, 0, \dots, 0), (0, 1, 0, \dots, 0), \dots, (0, \dots, 0, 1)\}$. The orthogonal matrixes H_A, H_B transform the basis of the spaces respectively into different coordinates. In this point of view, we can't make direct compare of row vectors between two parties. And the dot product of perturbed rows will be:

$$Y_A \cdot Y_B^T = X_A \cdot H_A \cdot H_B^T \cdot X_B^T \quad (5)$$

In order to get the original dot product from (5), the point is to know $H_A \cdot H_B^T$, which we don't directly provide and the reason will be deferred to the "Privacy Analysis" part.

Overall, for clients, our transmitting protocol consists of the following steps:

1. Generate an orthogonal matrix H using the algorithm in Section 3.2.
2. Transform the original data X into $Y = X \cdot H$, and send Y to miner.
3. Generate two "RD" matrix RD_1, RD_2 independently, whose $rd_{i,i} = 1$, $rd_{i,j} (i \neq j)$ are realizations of i.i.d. variables, i.e. with the same normal distribution $N(\mu, \delta^2)$ and mutually independent.
4. Receive the matrix from his former neighbor, right multiply it by $RD_1 \cdot H$ and send the product as the "perturbation matrix" to the miner. Send $(RD_2 \cdot H)^T$ to his latter neighbor.

What the miner gets includes the perturbed data matrixes from all parties and the "perturbation matrixes" e.g. $T_{A,B} = H_A^T \cdot RD_A^T \cdot RD_B \cdot H_B$ as the randomized transforming matrix from coordinate system A to B. When the miner is going to calculate the original $X_A \cdot X_B^T$, he uses $Y_A \cdot T_{A,B} \cdot Y_B^T$ to substitute for it. Before clustering, the miner should integrate all parties' data into one coordinate system. The way he achieves it consists of the following steps:

1. Receive the perturbed data sets from all n clients.
2. Determine the target coordinate system ("TCS" for short) by randomly designate one party's coordinate system.
3. If a row vector is not in "TCS", find the shortest perturbation matrix sequence from both transforming directions, and transform it into "TCS" by multiplying $T_{i,i+1} \cdot T_{i+1,i+2} \cdot \dots \cdot T_{TCS-1,TCS}$ or $T_{i,i-1} \cdot \dots \cdot T_{1,n} \cdot \dots \cdot T_{TCS+1,TCS}$.

Thus, the above steps of both client and miner make up of our multi-party privacy preserving clustering protocol. Then, we will analyze the accuracy level it reaches and the privacy it can preserve.

5 Accuracy Analysis

In the common orthogonal transformation, dot product operation is performed without losing accuracy, but has big security problem (to be discussed in next section). When the "RD" matrix takes part in, $X_A \cdot X_B^T$ changes to:

$$\begin{aligned} & Y_A \cdot (H_A^T \cdot RD_A^T \cdot RD_B \cdot H_B) \cdot Y_B^T \\ &= (X_A \cdot H_A) \cdot (H_A^T \cdot RD_A^T \cdot RD_B \cdot H_B) \cdot (H_A^T \cdot X_B^T) \\ &= X_A \cdot RD_A^T \cdot RD_B \cdot X_B^T. \end{aligned} \quad (6)$$

which brings variance to the result. Thus, the investigation of the characteristics of "RD" matrix products is necessary.

Lemma 2. *Assume that P, Q are two independent variables, $P \sim N(\mu_p, \delta_p^2)$, $Q \sim N(\mu_q, \delta_q^2)$. Then the variance of their product is the product of their variance, if they both have their means equal to 0.*

Proof. We calculate the expressions of expectation and variance as follows:

$$E(P_i \cdot Q_i) = E(P_i) \cdot E(Q_i) = \mu_p \cdot \mu_q \quad (7)$$

$$\begin{aligned} & D(P_i \cdot Q_i) \\ &= E(P_i^2 \cdot Q_i^2) - E(P_i \cdot Q_i)^2 \\ &= E(P_i^2) \cdot E(Q_i^2) - E(P_i)^2 \cdot E(Q_i)^2 \\ &= (D(P_i) + E(P_i)^2) \cdot (D(Q_i) + E(Q_i)^2) - E(P_i)^2 \cdot E(Q_i)^2 \\ &= D(P_i) \cdot D(Q_i) + E(P_i)^2 \cdot D(Q_i) + E(Q_i)^2 \cdot D(P_i) \end{aligned} \quad (8)$$

If we have $P \sim N(0, \delta_p^2)$, $Q \sim N(0, \delta_q^2)$, we will have $D(P_i \cdot Q_i) = \delta_p^2 \cdot \delta_q^2$. This is the most extreme condition that the variance is the smallest, even smaller than the variance of each variable if $\delta_p^2, \delta_q^2 < 1$. \square

Lemma 3. *Assume the recursive sequence $a_n = \sum_{i=1}^k p_i \cdot a_{n-i}$. If its characteristic equation $x^k = \sum_{i=1}^k p_i \cdot x^{k-i}$ has k different nonzero real roots $x_i (i=1, 2, \dots, k)$, then $a_n = \sum_{i=1}^k q_i \cdot x_i^n$, ($q_i = f(a_1, a_2, \dots, a_k)$) .*

Proof. For j th real root x_j , $x_j^k = \sum_{i=1}^k p_i \cdot x_j^{k-i}$. Since $x_j \neq 0$, we have

$$x_j^n = \sum_{i=1}^k p_i \cdot x_j^{n-i}, \quad (9)$$

Thus, $\{x_j^n\}$ meets the pattern of $\{a_n\}$.

For $\forall u, v (1 \leq u, v \leq k)$, $\forall s_1, s_2 \in \mathbb{R}$, we have:

$$s_1 \cdot x_u^n + s_2 \cdot x_v^n = \sum_{i=1}^k p_i \cdot (s_1 \cdot x_u^{n-i} + s_2 \cdot x_v^{n-i}) , \quad (10)$$

also meets the pattern of $\{a_n\}$, thus, we can derive: $a_n = \sum_{i=1}^k s_i \cdot x_i^n$, ($s_i \in \mathbb{R}$).

To determine s_i , we have $\begin{bmatrix} x_1 & x_2 & \cdots & x_k \\ x_1^2 & x_2^2 & \cdots & x_k^2 \\ \dots & \dots & \dots & \dots \\ x_1^k & x_2^k & \cdots & x_k^k \end{bmatrix} \cdot \begin{bmatrix} s_1 \\ s_2 \\ \dots \\ s_k \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_k \end{bmatrix}$, since x_i are different roots, the equation set has unique solution $\{s_1, s_2, \dots, s_k\}$. \square

Lemma 4. Suppose there are n parties, and each of them has its dataset of m columns. Let a_t, b_t represent the variance of the diagonal elements and non-diagonal elements in the matrix which is the result of t times multiplication of $m \times m$ "RD" matrixes. When these n parties use $\delta^2 = \frac{1}{r \cdot n \cdot m^2}$, ($r \in \mathbb{R}$) to generate the "RD" matrixes, the maximum variance of the dot product between data records varies inversely as r , and directly as the product of the average squared value of record vector entries.

Proof. By Lemma 2, we derive from Section 4 the variance of the entries in the final perturbation matrix as:

$$a_t = a_{t-1} + (m-1)b_{t-1} \cdot \delta^2, \quad b_t = b_{t-1} + (a_{t-1} + 1) \cdot \delta^2 + (m-2)b_{t-1} \cdot \delta^2 . \quad (11)$$

Let $\alpha = (m-1) \cdot \delta^2, \beta = 1 + (m-2) \cdot \delta^2$, Then

$$b_t = \delta^2 \cdot a_{t-1} + \beta \cdot b_{t-1} + \delta^2 \quad (12)$$

$$\begin{aligned} a_t &= a_{t-1} + \alpha \cdot (\delta^2 \cdot a_{t-2} + \beta \cdot b_{t-2} + \delta^2) \\ &= (1 + \beta) \cdot a_{t-1} + (\alpha \cdot \delta^2 - \beta) a_{t-2} + \alpha \cdot \delta^2 . \end{aligned} \quad (13)$$

From above, we can derive:

$$a_t - a_{t-1} = (1 + \beta) \cdot (a_{t-1} - a_{t-2}) + (\alpha \cdot \delta^2 - \beta)(a_{t-2} - a_{t-3}) . \quad (14)$$

The corresponding characteristic equation for $a_t - a_{t-1}$ is:

$$x^2 - (1 + \beta) \cdot x - (\alpha \cdot \delta^2 - \beta) = 0 . \quad (15)$$

Hence:

$$x_{1,2} = \frac{1 + \beta \pm \sqrt{(\beta - 1)^2 + 4\alpha \cdot \delta^2}}{2} \quad (m > 1, \delta^2 > 0, \text{thus } x_1 \neq x_2) . \quad (16)$$

By Lemma 3, we can derive the expressions:

$$\begin{cases} a_t - a_{t-1} = c_1 \cdot x_1^t + c_2 \cdot x_2^t \\ b_t = \frac{a_{t+1} - a_t}{\alpha} = \frac{c_1 \cdot x_1^{t+1} + c_2 \cdot x_2^{t+1}}{\alpha} \end{cases} \quad (c_1, c_2 \in \mathbb{R}) \quad (17)$$

$$\begin{cases} c_1 = \frac{x_2 \cdot \alpha \cdot b_1 - a_3 + a_2}{x_2 \cdot x_1^2 - x_1^3} = \frac{x_2 \cdot \alpha \cdot b_1 - (m-1) \cdot b_2 \cdot \delta^2}{x_2 \cdot x_1^2 - x_1^3} \\ c_2 = \frac{x_1 \cdot \alpha \cdot b_1 - a_3 + a_2}{x_1 \cdot x_2^2 - x_2^3} = \frac{x_1 \cdot \alpha \cdot b_1 - (m-1) \cdot b_2 \cdot \delta^2}{x_1 \cdot x_2^2 - x_2^3} \end{cases} \quad (18)$$

Combining (17) and (18), we have:

$$\begin{aligned} b_t &= \frac{c_1 \cdot x_1^{t+1} + c_2 \cdot x_2^{t+1}}{\alpha} \\ &= \frac{x_2 \cdot \alpha \cdot b_1 - (m-1) \cdot b_2 \cdot \delta^2}{x_2 \cdot x_1^2 - x_1^3} \cdot \frac{x_1^{t+1}}{\alpha} + \frac{x_1 \cdot \alpha \cdot b_1 - (m-1) \cdot b_2 \cdot \delta^2}{x_1 \cdot x_2^2 - x_2^3} \cdot \frac{x_2^{t+1}}{\alpha} \\ &\approx \frac{1}{m} \cdot \sum_{i \geq 1} C_n^i [m^i - (-1)^i] \cdot \delta^{2i} \\ &\leq \frac{e}{m^2 r} \quad (\text{if } \delta^2 = 1/rnm^2) . \end{aligned} \quad (19)$$

The longest path of "perturbation matrix" to "TCS" is $n-1$ when there are n participants. Since the miner chooses the shortest transformation sequence, and we transform all parties to one "TCS", the largest number of "RD" matrixes in transformation path between vectors is n .

When $\delta^2 = 1/rnm^2$, we have $b_t \leq \frac{e}{m^2 r}$. The dot product of row vectors in data matrix $x_A \cdot RD^n \cdot x_B^T$, comparing with the original $x_A \cdot x_B^T$:

$$\begin{aligned} &Var((x_A \cdot x_B^T) - (x_A \cdot RD^n \cdot x_B^T)) \\ &\leq \frac{e}{m^2 r} \cdot \sum_{1 \leq i, j \leq m} ((x_A)_i \cdot (x_B)_j)^2 \\ &= \frac{e}{r} \cdot E((x_A)_i^2) \cdot E((x_B)_j^2) . \end{aligned} \quad (20)$$

According to (20), the choice of parameter r has to respect the maximum variance of squared entries in row vectors, so that it will relieve the impact by those entries having greater departure. \square

6 Privacy Analysis

6.1 Semi-honest Condition

The difficulty to infer the original attribute X through the perturbed Y is measured by $Var(Y - X)$, and the privacy analysis of orthogonal transformation has been discussed in [9]. In the semi-honest conditions, assume no malicious adversaries; we use the variable-wised privacy transformation matrix (algorithm in Section 3.2), so that each variable can be protected according to its own importance specified by its owner.

6.2 Malicious Condition

While common orthogonal matrix without perturbation seems to work fine in the semi-honest conditions, there will be severe problems in the malicious model.

Problems in Common Way. For example, in the two-party (A, B) situation, when the miner receives $X_A \cdot H_A, X_B \cdot H_B, H_A^T \cdot H_B$, he himself can only derive $X_A \cdot H_B, X_B \cdot H_A$, which do not improve his ability to probe A and B .

However, conspiring with a certain client makes orthogonal transformation have no resistances. Suppose party A colludes with the miner, he shares his H_A with his conspirator, then the miner immediately knows H_B through multiplying H_A by $H_A^T \cdot H_B$. Right away, B 's data will be disclosed.

Furthermore, it is not the worst situation. If the "TCS" is set to be the same as A 's coordinate system, with which all the other parties have to comply, the miner gets $H_{party_1}^T \cdot H_A, H_{party_2}^T \cdot H_A, \dots$. Once A betrays, it can be a disaster.

Problems of Direct Randomization. Using randomization to stop conspiracy is a direct way. Moreover, we will also illustrate that the singular value decomposition (SVD) method is a grave threat to data security in the common randomization form.

It's straightforward to achieve higher security level by adopting randomized diagonal matrix ("RanDiag" for short) as perturbation matrix, in which all diagonal elements are i.i.d. from normal distribution. Consequently, every element is perturbed by a diagonal element. Thus, the cumulative effects of several "RanDiag" can be obvious: the final effect is also a diagonal matrix, every original data column is impacted by one of its element, and each element of the result "RanDiag" $randiag_{i,i}$ depends on the entries of the same position in each multiplier. Moreover, due to the i.i.d. characteristic, the expectation and the variance of $randiag_{i,i}$ can be obtained according to Lemma 2.

Generally speaking, the "RanDiag" hides H_B behind $Randiag \cdot H_B$ in the former example, and therefore seems protecting B from complicity; also in the meantime keeping accuracy in some extents. However, if the miner applies SVD on the transforming matrix $H_A^T \cdot Randiag_A \cdot Randiag_B \cdot H_B$, he will get $(Randiag_A \cdot Randiag_B)$ with exact elements, but not in the correct order. Taken in this sense, SVD can improve the probability of revealing B 's data into $\frac{1}{O(\text{number of columns})}$, in case A conspires with miner.

Privacy Levels of Our Method. Our method tries to avoid the above problems, we let neighbor parties send out the product with their "RD" matrix, and i.e. every party's transformation matrix is randomized by a "RD" matrix. Like the "RanDiag's" process, the miner gets $H_A^T \cdot RD_A^T \cdot RD_B \cdot H_B, X_A \cdot H_A$ and $X_B \cdot H_B$. Comparing with the common orthogonal transformation way leaking the original X_B , the conspirators get $X_B \cdot RD_B$ instead. Every entry in the original vector contributes its own part to protect each of them in the perturbed version:

$$y_i = x_i + \sum_{j \neq i} r d_{j,i} \cdot x_j \quad (x_i, y_i : \text{the } i\text{th element in the row vectors of } X, Y). \quad (21)$$

Since $r d_{j,i}$ ($j \neq i$) is i.i.d. and generated by $N(0, \delta^2)$, then $(\sum_{j \neq i} (r d_{j,i} \cdot x_j)) \sim N(0, \delta^2 \cdot \sum_{j \neq i} x_j^2)$.

Thus, by Lemma 4, we have:

$$\text{Var}(y_i - x_i) = \delta^2 \cdot \sum_{j \neq i} x_j^2 = \sum_{j \neq i} \frac{x_j^2}{r \cdot m^2 \cdot n} \leq \frac{E(x_j^2)}{r \cdot m \cdot n}. \quad (22)$$

Therefore, in the face of complicity, our method may not maintain the variable order of privacy levels as user specified, but we will have the interval width $6.8\sqrt{E(x_j^2)/(r \cdot m \cdot n)}$ define the amount of privacy at about 100% confidence level [1]. It can be inferred from (19)(22) that the privacy level varies inversely as the average length of the matrix paths, thus, choosing the shortest path to "TCS" for every participant is necessary.

7 Experiment Result

The dataset "Synthetic Control Chart Time Series" are obtained from the UCI KDD Archive [17]. It has 60 columns with similar averages and 600 rows ($\text{Var}(E(\text{column}_i))/E(E(\text{column}_i)) = 1.9\%$). The clustering result is shown in Figure 1(a) with 6 clusters. We horizontally partition the data in order to simulate different parties. Then, for each party, we generate a random orthogonal matrix, and a "RD" matrix with $\delta^2 = 1/r(60)^2n$. In this phase, the orthogonality of the transformation matrix is enough, so we generate random orthogonal matrixes for every simulated party. In the experiment, we assume all parties have the same parameter r in each turn, and we compare its effect on the accuracy. This can tell us the lowest accuracy level under maximum δ .

In order to simulate the miner using k-means for clustering, we randomly choose a party in every turn to be the "TCS", find a shortest path to it for other parties to transform to, and link the transformation matrixes on this path by multiplication. With respect to the clustering result of original data, we calculate the error rate for clustering as $\frac{\text{number of records in wrong clusters}}{\text{number of all records}} \cdot 100\%$. During the experiment, we iterate 10 times in every combination of parameters to get the average error rate. For the k-means clustering algorithm, we repeat the process 10 times to choose the result with the smallest sum of distances.

As shown in Figure 2, we test the impact on clustering result with combinations of party number and r . The error rate increases slowly with the number of parties, because the variance of calculation is closer to the maximum variance which is determined by r and m , as n gets larger. Also, the bigger $1/r$ is, the more the maximum variance. We also give a directviewing of the clustering result in Figure 1(b). Comparing with Figure 1(a), we can also find out that due to the orthogonal transformation keeping the distances among data vectors, the 6 clusters vary little.

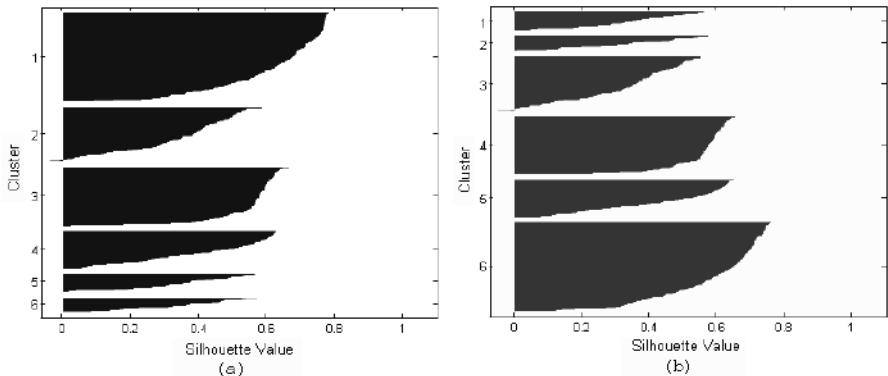


Fig. 1. (a)Cluster silhouettes for the original data. (b)Cluster silhouettes for the perturbed dataset where $n = 10, r = 3$.

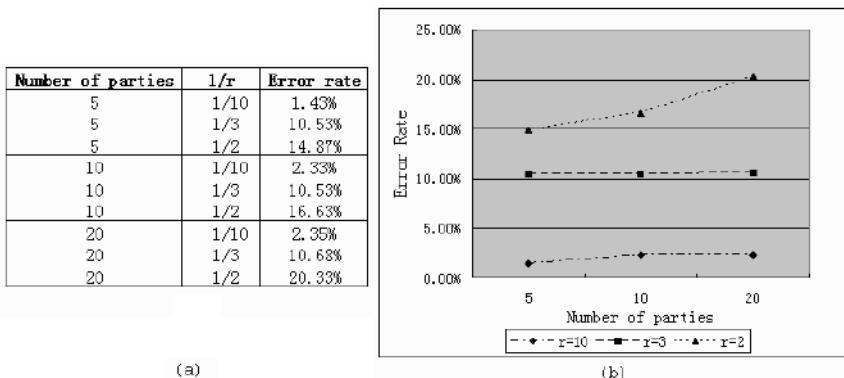


Fig. 2. (a)K-Means clustering with different combinations of parameters.
(b)Comparison of clustering accuracy.

8 Conclusions

In this paper, we have introduced a multi-party clustering method using "RD" matrix for protecting the original records both in semi-honest and malicious conditions. The "RD" matrix leverages orthogonal transformation in the semi-honest condition to avoid the compromise between privacy and accuracy, and also protects data from malicious complicity by randomization without losing much accuracy. As the increasing of participants, the influence to the privacy loss is within linear level. We've also presented an efficient algorithm to achieve the attribute-wised privacy concerns.

The orthogonal-like method is a promising way to protect the sensitive data in some mining tasks. Although it has limitations, combining it with other

techniques may accomplish more tasks. We regard our work as an initial step. Further research will include more work in the malicious model and extending this idea to other data mining algorithms.

References

1. Agrawal, R., S.R.: Privacy-preserving data mining. In: Proc. of the ACM SIGMOD Conference on Management of Data. (2000)
2. Lindell, Y., P.B.: Privacy preserving data mining. In: Proc. of the 20th Annual International Cryptology Conference on Advances in Cryptology. (2000)
3. Rizvi, S., H.J.: Maintaining data privacy in association rule mining. In: Proc. of the 28th Conference on Very Large Data Bases. (2002)
4. Evfimievski, A., S.R.A.R.G.J.: Privacy preserving mining of association rules. *Information Systems* **29**(4) (2004) 343–364
5. Xia, Y., Y.Y.C.Y.: Mining association rules with non-uniform privacy concerns. In: Proc. of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery. (2004)
6. Du, W., Z.Z.: Using randomized response techniques for privacy-preserving data mining. In: Proc. of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining. (2003)
7. Warner, S.: Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association* **60** (1965) 63–69
8. Oliveira, S., Z.O.: Privacy preserving clustering by data transformation. In: Proc. of the 18th Brazilian Symposium on Databases. (2003)
9. Chen, K., L.L.: Privacy preserving data classification with rotation perturbation. In: Proc. of the 5th IEEE International Conference on Data Mining. (2005)
10. Liu, K., K.H.R.J.: Random projection-based multiplicative data perturbation for privacy preserving distributed data mining. *IEEE Transactions on Knowledge and Data Engineering* **18**(1) (2006) 92–106
11. Vaidya, J., C.C.: Privacy-preserving k-means clustering over vertically partitioned data. In: Proc. of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining. (2003)
12. Vaidya, J., C.C.: Privacy preserving association rule mining in vertically partitioned data. In: Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. (2002)
13. Kantarcioglu, M., C.C.: Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering* **16**(9) (2004) 1026–1037
14. Wright, R., Y.Z.: Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In: Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. (2004)
15. Agrawal, D., A.C.: On the design and quantification of privacy preserving data mining algorithms. In: Proc. of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. (2001)
16. Stewart, G.: The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM Journal on Numerical Analysis* **17**(3) (1980) 403–409
17. Hettich, S., B.S.: The uci kdd archive. Univeristy of California, Irvine, Department of Information and Computer Science (1999)

Privacy-Preserving Frequent Pattern Sharing*

Zhihui Wang¹, Wei Wang¹, Baile Shi¹, and S.H. Boey²

¹ Department of Computing and Information Technology,
Fudan University, Shanghai, 200433, China

{zhhwang, weiwang1, bshi}@fudan.edu.cn

² Gentec Pte Ltd, 80 Genting Lane #04-07, Singapore, 349565
shboey@gintic.com

Abstract. Some of the knowledge discovered by data mining may contain sensitive information, which should be hidden before sharing the result of data mining. In this paper, we consider that the knowledge for sharing is discovered by frequent pattern mining, and some of the frequent patterns are private, which cannot be shared. Our problem of privacy-preserving frequent pattern sharing is to hide these private patterns before sharing the result of frequent pattern mining, and at the same time maximize the number of non-private frequent patterns to be shared. We show that this problem is NP-hard, and present three item-based pattern sanitization algorithms for transforming the result of frequent pattern mining into a privacy-free frequent pattern set.

Keywords: Frequent pattern, private pattern, privacy preservation.

1 Introduction

Data mining can help to discover useful knowledge from large amount of data. In many applications, we need to share the discovered knowledge by data mining with others. For example, two retailers *A* and *B* cooperate to discover the purchase behaviors of their customers. Due to legal prohibitions, they cannot share their transaction database directly. One way for solving this problem is that each retailer can first perform frequent pattern mining on its own database. Then, each retailer can exchange their mining results with the other. The authors in [12] have shown that the expensive computational cost makes it impractical to reconstruct the original database from the set of shared frequent patterns. Therefore, retailer *A* (or *B*) will not have the danger of disclosing its original data when it shares its local frequent patterns with the other.

However, the knowledge discovered by data mining may also contain sensitive information. The disclosure of sensitive information can bring threats against personal privacy, commercial secret, and even national security [3]. Consider retailer *A* in the above example. Assume that after performing frequent pattern

* This research was supported by the Shanghai Rising-Star Program (No. 05QMX1405), the National Natural Science Foundation of China (No. 60303008), the National Grand Fundamental Research 973 Program of China (No. 2005CB321905).

mining on its own data, retailer A finds that its customers tend to purchase merchandise x and y at the same time. Retailer A regards this knowledge as sensitive information, and does not want to disclose it to retailer B . This is because with this knowledge, B may launch sales promotion and offer a lower price for customers who buy x and y together. Thus, A will face the danger of losing its customers.

In this paper, we address the problem of privacy-preserving frequent pattern sharing. The result of frequent pattern mining is the collection of all frequent patterns with their supports. Frequent patterns are very useful in many areas, e.g., association rule discovery, feature extraction, classification, and clustering [4]. The scenario that we consider is as follows: given the result of frequent pattern mining, some of the discovered frequent patterns are private and cannot be shared. Our problem of privacy-preserving frequent pattern sharing is to protect private patterns from disclosure, while maximizing the number of non-private frequent patterns to be shared.

The main contributions in this paper are as follows. First, we present the notation of *privacy-free frequent pattern set*. A privacy-free frequent pattern set is a subset of the result of frequent pattern mining, and can be shared without disclosing sensitive information. Besides maintaining privacy, a privacy-free frequent pattern set has other advantages. It does not contain any fake frequent pattern, which does not appear in the original database. Also, the support of any frequent pattern in a privacy-free frequent pattern set is the same as that in the original database. Second, we show that finding an optimal solution to our problem of privacy-preserving frequent pattern sharing is NP-hard. Our proof is based on a reduction from the hitting set problem [5]. Third, we present a framework, called *item-based pattern sanitization*, and provide three pattern sanitization algorithms. Each of the algorithms can guarantee transforming the result of frequent pattern mining into a privacy-free frequent pattern set.

This paper is organized as follows. In the next section, we review the related work in literature. In Section 3, we define our problem of privacy-preserving frequent pattern sharing, and present the notation of privacy-free frequent pattern set. We also show that finding an maximal privacy-free frequent pattern set to our problem is NP-hard. In Section 4, we present the framework of item-based pattern sanitization, and give three pattern sanitization algorithms. Each of them can guarantee generating a privacy-free frequent pattern set. We evaluate the performance of our algorithms in Section 5. Finally, we conclude our work in Section 6.

2 Related Work

In recent years, many efforts have been made to address the problem of privacy-preserving data mining. The studies closely related to our work are as follows.

Oliveira et al. investigated security issues of association rule sharing in [6]. Given a set R of association rules for sharing, a subset R_R of R is restricted from disclosure. They converted association rules into the corresponding frequent

patterns, then sanitized the set of frequent patterns by an algorithm, called *Downright Sanitizing Algorithm* (DSA). However, DSA has its limitation. It cannot always prevent privacy breach completely. An attacker can still infer sensitive information from DSA's sanitization result [7].

Atzori et al. [8] considered the anonymity issue raised by frequent patterns sharing. That is, given a set \mathcal{F} of frequent patterns for sharing, and an anonymity threshold k ($k \in \mathbb{N}$), if it can be inferred from \mathcal{F} that there exists a pattern p , and the support count of p is less than k , then sharing \mathcal{F} may pose threat to individual's anonymity. The scenario we considered is completely different from their studies. We consider that some of the discovered frequent patterns themselves are private, and need to be hidden when sharing the result of frequent pattern mining.

There are also another research direction related to our work. Instead of sharing the knowledge discovered by data mining, it considers to share a dataset directly. Before sharing, the original dataset is transformed so that no private pattern or rule can be mined from the resultant dataset. This procedure is called *data sanitization*. Atallah et al. [9] proved that the problem of finding an optimal solution for data sanitization is NP-hard. Different heuristic approaches for data sanitization were proposed in [9][10][11][12]. One main disadvantage of data sanitization is that it will change the supports of non-private patterns, which may not be tolerable in many applications.

3 Problem Statement

3.1 Basic Concepts and Notations

For facilitating our discussion, we first introduce some basic concepts and notations. Let $\mathcal{I} = \{i_1, i_2, \dots, i_n\}$ be a set of literals, called *items*. A transaction T is a set of items from \mathcal{I} , i.e., $T \subseteq \mathcal{I}$, and a transaction database \mathcal{D} is a set of transactions.

A *pattern* (or *itemset*) p is a set of items from \mathcal{I} , i.e., $p \in 2^{\mathcal{I}}$. If there are k items in p , we say that the length of p is k , denoted by $|p| = k$. For brevity, sometimes we write a pattern $p = \{i_j, i_{j+1}, \dots, i_k\}$ in the form of $p = i_j i_{j+1} \dots i_k$.

Definition 1. A transaction T contains pattern p if and only if $p \subseteq T$. In a transaction database \mathcal{D} , the support of pattern p is the fraction of transactions in \mathcal{D} that contain p , denoted by $\text{sup}(p)$.

Definition 2. Given transaction database \mathcal{D} and minimum support threshold σ ($0 < \sigma < 1$), we say that p is a σ -frequent pattern of \mathcal{D} if $\text{sup}(p) \geq \sigma$ in \mathcal{D} . Particularly, if $|p| = 1$, we also say that p is a σ -frequent item of \mathcal{D} .

For brevity, we sometimes call σ -frequent pattern frequent pattern if the context is clear. Given a transaction database \mathcal{D} , and minimum support threshold σ , the task of frequent pattern mining (or frequent itemset mining) is to find the

collection of all σ -frequent patterns with their supports in \mathcal{D} . There are many typical approaches for frequent pattern mining, e.g., Apriori, FP-growth [4]. We denote the result of frequent pattern mining with $\mathcal{F}(\mathcal{D}, \sigma)$ (or \mathcal{F} in short if the context is clear), where $\mathcal{F}(\mathcal{D}, \sigma) = \{(p, \text{sup}(p)) | \text{sup}(p) \geq \sigma\}$.

3.2 The Problem

In this paper, we consider that the data are stored in a transaction database, and the knowledge is discovered by frequent pattern mining, and represented in the form of a set of frequent patterns with their supports. Some of the frequent patterns contain sensitive information, which cannot be disclosed. We called these patterns *private patterns*. Generally, whether a pattern is private or not is determined by the data owner after performing frequent pattern mining on the original database.

Our problem of privacy-preserving frequent pattern sharing can be stated formally as follows: given the result \mathcal{F} of frequent pattern mining for sharing, let $\mathcal{P}_s \subset \mathcal{F}$ be the set of private patterns, our problem consists in transforming \mathcal{F} into \mathcal{F}' , which must satisfy the following conditions: (1) $\mathcal{F}' \subseteq \mathcal{F} - \mathcal{P}_s$; (2) $\forall q \in \mathcal{P}_s : q \not\subseteq \mathcal{I}_{\mathcal{F}'}$, where $\mathcal{I}_{\mathcal{F}'} = \{i | i \in p \wedge p \in \mathcal{F}'\}$; (3) $\forall \mathcal{F}''$ satisfying (1) and (2), $|\mathcal{F}''| \leq |\mathcal{F}'|$.

We assume $|\mathcal{P}_s| \ll |\mathcal{F}|$, where $|\mathcal{P}_s|$ and $|\mathcal{F}|$ are the number of private patterns and all σ -frequent patterns, respectively. We also assume that what an adversary can get is only \mathcal{F}' , i.e., the set of shared σ -frequent patterns with their supports. We do not consider an adversary having other background knowledge.

Given a solution \mathcal{F}' to our problem of privacy-preserving frequent pattern sharing, the above first condition requires that \mathcal{F}' does not contain any private patterns. Furthermore, it requires that \mathcal{F}' is a part of the knowledge discovered by frequent pattern mining. That is, there is no any fake σ -frequent pattern in \mathcal{F}' . More importantly, for any shared pattern $p \in \mathcal{F}'$, $\text{sup}(p)$ cannot be changed, which further make sure that the accuracy of shared knowledge is maintained.

The second condition ensures that any private pattern is hidden completely by ruling out any chance to infer the existence of private pattern from \mathcal{F}' . If $q \subseteq \mathcal{I}_{\mathcal{F}'}$, an adversary may guess that q appears in the original database, then the adversary can have the chance to infer the existence of q in the original database with some technologies of inverse mining attack. By constraining $q \not\subseteq \mathcal{I}_{\mathcal{F}'}$, we try to prevent an adversary from this kind of guess.

In the third condition, $|\mathcal{F}'|$ and $|\mathcal{F}''|$ are the number of σ -frequent patterns in \mathcal{F}' and \mathcal{F}'' , respectively. This condition states that we also expect to find an optimal solution \mathcal{F}' to our problem. That is, we want to maximize the number of non-private patterns in \mathcal{F}' .

Definition 3. *Given a set \mathcal{F}' of frequent patterns, if \mathcal{F}' meets the first and the second conditions in the problem of privacy-preserving frequent pattern sharing, we call \mathcal{F}' privacy-free frequent pattern set. If a privacy-free frequent pattern set \mathcal{F}' also meets the third condition, that is, it is maximal, we called it maximal privacy-free frequent pattern set.*

We expect to generate an maximal privacy-free frequent pattern set for our problem of privacy-preserving frequent pattern sharing. Unfortunately, finding an optimal solution to our problem is NP-hard.

Theorem 1. *Find an optimal solution to our problem of privacy-preserving frequent pattern sharing is NP-hard.*

Proof. We prove the theorem by a reduction from the problem of hitting set [5]. Given a set $C \subseteq 2^S$, where S is a finite set, the problem of hitting set is to find a smallest subset S' of S such that each $c \in C$ contains at least one element in S' . In this proof, we assume that each $c \in C$ consists of exactly two elements in S . Note that the hitting set problem remains NP-hard under this assumption.

We can create an instance of our problem of privacy-preserving frequent pattern sharing in terms of the above hitting set problem in polynomial time as follows: “Let $\mathcal{I}_C = \{i | i \in c \wedge c \in C\}$. Suppose database $\mathcal{D} = C \cup \mathcal{D}_C$, where $\mathcal{D}_C = \{\{i\} | i \in \mathcal{I}_C\}$, and minimum support threshold $\sigma = 1/n$, where n is the number of transactions in \mathcal{D} , the collection of all σ -frequent patterns in \mathcal{D} will be $\mathcal{F} = C \cup \mathcal{D}_C$. Let $\mathcal{P}_s = C$ be a set of private patterns.”

Suppose \mathcal{F}' is an optimal solution to the above problem of privacy-preserving frequent pattern sharing. That is, \mathcal{F}' is an maximal privacy-free frequent pattern set. We can show that $\mathcal{I}_C - \mathcal{I}_{\mathcal{F}'}$ is a solution to the corresponding hitting set problem, where $\mathcal{I}_{\mathcal{F}'} = \{i | i \in p \wedge p \in \mathcal{F}'\}$. The details are as follows:

- (1) Since $C \subseteq 2^S$, we have $\mathcal{I}_C \subseteq S$. Therefore, $\mathcal{I}_C - \mathcal{I}_{\mathcal{F}'}$ is a subset of S .
- (2) Since $\mathcal{P}_s = C$ and \mathcal{F}' is an maximal privacy-free frequent pattern set, we have $\forall c \in C : c \not\subseteq \mathcal{I}_{\mathcal{F}'}$. Because also $\forall c \in C : c \subseteq \mathcal{I}_C$, we have $c \cap (\mathcal{I}_C - \mathcal{I}_{\mathcal{F}'}) \neq \emptyset$. That is, each $c \in C$ contains at least one element in $\mathcal{I}_C - \mathcal{I}_{\mathcal{F}'}$.
- (3) Suppose that S' is any solution to the hitting set problem, then we have $S' \subseteq \mathcal{I}_C$. Next, we will show that $|S'| \geq |\mathcal{I}_C - \mathcal{I}_{\mathcal{F}'}|$ by contradiction. Assume that $|S'| < |\mathcal{I}_C - \mathcal{I}_{\mathcal{F}'}|$. Let $\mathcal{I}_{\tilde{\mathcal{F}}} = \mathcal{I}_C - S'$ and $\tilde{\mathcal{F}} = \{\{i\} | i \in \mathcal{I}_{\tilde{\mathcal{F}}}\}$, we have $\tilde{\mathcal{F}} \subseteq \mathcal{D}_C$. Notice that $\mathcal{D}_C = \mathcal{F} - \mathcal{P}_s$, we have $\tilde{\mathcal{F}} \subseteq \mathcal{F} - \mathcal{P}_s$. Because also $\mathcal{P}_s = C$, we have $\forall q \in \mathcal{P}_s : |q| = 2$. So $\forall q \in \mathcal{P}_s : q \not\subseteq \mathcal{I}_{\tilde{\mathcal{F}}}$. Hence, $\tilde{\mathcal{F}}$ is a privacy-free frequent pattern set to our problem of privacy-preserving frequent pattern sharing. According to our assumption $|S'| < |\mathcal{I}_C - \mathcal{I}_{\mathcal{F}'}|$, we have $|\mathcal{I}_C - \mathcal{I}_{\tilde{\mathcal{F}}}| < |\mathcal{I}_C - \mathcal{I}_{\mathcal{F}'}|$, then $|\mathcal{I}_{\tilde{\mathcal{F}}}| > |\mathcal{I}_{\mathcal{F}'}|$. Since $\mathcal{F} = C \cup \mathcal{D}_C$ and $\mathcal{P}_s = C$, we know that $\mathcal{F}' = \{\{i\} | i \in \mathcal{I}_{\mathcal{F}'}\}$ and thus $|\mathcal{F}'| = |\mathcal{I}_{\mathcal{F}'}|$. Because $\tilde{\mathcal{F}} = \{\{i\} | i \in \mathcal{I}_{\tilde{\mathcal{F}}}\}$, we have $|\tilde{\mathcal{F}}| = |\mathcal{I}_{\tilde{\mathcal{F}}}|$. Therefore, $|\tilde{\mathcal{F}}| > |\mathcal{F}'|$. This conflicts with the condition that \mathcal{F}' is an maximal privacy-free frequent pattern set.

Therefore, $\mathcal{I}_C - \mathcal{I}_{\mathcal{F}'}$ is a solution to the hitting set problem. \square

4 Generating Privacy-Free Frequent Pattern Set

In this section, we first give a framework, called *item-based pattern sanitization*, which can guarantee transforming \mathcal{F} to a privacy-free frequent pattern set \mathcal{F}' . Then, based on our framework, we present three heuristic algorithms for privacy-preserving frequent pattern sharing.

4.1 The Framework of Item-Based Pattern Sanitization

Our framework of item-based pattern sanitization can be described as follows: “Given \mathcal{F} as the collection of all frequent patterns discovered by frequent pattern mining, let $\mathcal{P}_s \subset \mathcal{F}$ be a set of private patterns, we first initialize $\mathcal{F}' = \mathcal{F}$. Then, for each private pattern $q \in \mathcal{P}_s$, we choose an item $x \in q$ with some strategy, remove all the frequent patterns containing item x from \mathcal{F}' , and also all the private patterns containing item x from \mathcal{P}_s . This procedure is iterated until $\mathcal{P}_s = \emptyset$.” For facility, we call item x the *victim* of private pattern q .

Theorem 2. *Our framework of item-based pattern sanitization can generate a privacy-free frequent pattern set.*

Proof. From the procedure of our framework of item-based pattern sanitization, we know that for any $q \in \mathcal{P}_s$, there exists item $x \in q$ such that $\forall p \in \mathcal{F}' : p \not\ni x$. Hence, we have $\mathcal{F}' \wedge \mathcal{P}_s = \emptyset$ and $\forall q \in \mathcal{P}_s : q \not\subseteq \mathcal{I}_{\mathcal{F}'}$, where $\mathcal{I}_{\mathcal{F}'} = \{i | i \in p \wedge p \in \mathcal{F}'\}$. Because \mathcal{F}' is initialized as \mathcal{F} , we also have $\mathcal{F}' \subseteq \mathcal{F}$. Therefore, $\mathcal{F}' \subseteq \mathcal{F} - \mathcal{P}_s$. From Definition 3, we know that \mathcal{F}' is a privacy-free frequent pattern set. \square

4.2 The Sanitization Algorithms

Based on the framework proposed in Section 4.1, we give three pattern sanitization algorithms for privacy-preserving frequent pattern sharing in this section. They transform the result of frequent pattern mining into a privacy-free frequent pattern set with different strategies for choosing victims of private patterns.

Algorithm: RANDIPS

Input: frequent pattern set \mathcal{F} , private pattern set $\mathcal{P}_s (\mathcal{P}_s \subset \mathcal{F})$

Output: privacy-free frequent pattern set \mathcal{F}'

Method:

- 1: $\mathcal{F}' = \mathcal{F}$;
 - 2: **while** $\mathcal{P}_s \neq \emptyset$ **do begin**
 - 3: $S_c = \emptyset$; // S_c : a set of candidate items
 - 4: **for** each private pattern $q \in \mathcal{P}_s$ **do**
 - 5: **for** each item $i \in q$ and $i \notin S_c$ **do** $S_c = S_c \cup \{i\}$;
 - 6: select item $x \in S_c$ randomly as victim;
 - 7: $\mathcal{F}' = \mathcal{F}' - \{p | (x \in p) \wedge (p \in \mathcal{F}')\}$;
 - 8: $\mathcal{P}_s = \mathcal{P}_s - \{q | (x \in q) \wedge (q \in \mathcal{P}_s)\}$;
 - 9: **end**
-

Fig. 1. Algorithm RANDIPS

The first sanitization algorithm, called RANDIPS (RANDom Item-based Pattern Sanitization), is a naive algorithm, just for the purpose of performance comparison in Section 5. The details of RANDIPS are shown in Fig. 1. Given the collection \mathcal{F} of all frequent patterns discovered by frequent pattern mining, and

$\mathcal{P}_s \subset \mathcal{F}$ is a set of private patterns, RANDIPS first initializes $\mathcal{F}' = \mathcal{F}$. While \mathcal{P}_s is not empty, RANDIPS scans \mathcal{P}_s to construct a set of candidate items S_c at line 3 – 5. Then, it chooses randomly an item x as victim from S_c , and removes any frequent patterns containing x from \mathcal{F}' at line 7, any private patterns containing x from \mathcal{P}_s at line 8. When $\mathcal{P}_s = \emptyset$, we know from Theorem 2 that \mathcal{F}' is a privacy-free frequent pattern set.

Our second algorithm, called MINCIPS (MINimal Counter Item-based Pattern Sanitization), chooses victim with a greedy strategy during the course of pattern sanitization. The details of MINCIPS are shown in Fig. 2. For each item $i \in S_c$, MINCIPS maintains a counter $f cnt[i]$, which records the number of frequent patterns in \mathcal{F}' containing item i . Each time MINCIPS chooses from S_c the item x with the minimal counter $f cnt[x]$ as victim at line 7. By choosing such a victim, we expect to eliminate less number of frequent patterns from the result of frequent pattern mining. Similarly, according to Theorem 2, it is easy to show that MINCIPS generates a privacy-free frequent pattern set.

Algorithm: MINCIPS

Input: frequent pattern set \mathcal{F} , private pattern set $\mathcal{P}_s (\mathcal{P}_s \subset \mathcal{F})$

Output: privacy-free frequent pattern set \mathcal{F}'

Method:

```

1:  $\mathcal{F}' = \mathcal{F};$ 
2: while  $\mathcal{P}_s \neq \emptyset$  do begin
3:    $S_c = \emptyset;$ 
4:   for each private pattern  $q \in \mathcal{P}_s$  do
5:     for each item  $i \in q$  and  $i \notin S_c$  do  $S_c = S_c \cup \{i\}; f cnt[i] = 0;$ 
6:     for each  $p \in \mathcal{F}'$  do  $\forall i \in (p \cap S_c), f cnt[i] = f cnt[i] + 1;$ 
7:   select item  $x \in S_c$  such that  $x = \arg \min_{i \in S_c} \{f cnt[i]\};$ 
8:    $\mathcal{F}' = \mathcal{F}' - \{p | (x \in p) \wedge (p \in \mathcal{F}')\};$ 
9:    $\mathcal{P}_s = \mathcal{P}_s - \{q | (x \in q) \wedge (q \in \mathcal{P}_s)\};$ 
10: end

```

Fig. 2. Algorithm MINCIPS

We notice that there are often common items among multiple private patterns. For example, Suppose $q_1 = ab$ and $q_2 = ac$ are private patterns. If we choose item a as victim to sanitize frequent patterns, we can prevent the disclosure of both q_1 and q_2 at the same time. Incorporating this idea, we give an improved algorithm, called MAXSIPS (MAXimal Score Item-based Pattern Sanitization). The details of MAXSIPS are shown in Fig. 3. For each item $i \in S_c$, MAXSIPS maintains not only the counter $f cnt[i]$ as MINCIPS does, but also another counter $p cnt[i]$. The counter $p cnt[i]$ records the number of private patterns in \mathcal{P}_s containing item i . Furthermore, MAXSIPS calculates a score $s[i] = p cnt[i]/f cnt[i]$ for each item $i \in S_c$ at line 9, and chooses from S_c the item x with the maximal score $s[x]$ as victim at line 10. By choosing such a victim, we also expect to maximize the number of private patterns being hidden in an iteration, and thus reduce the

Algorithm: MAXSIPS

Input: frequent pattern set \mathcal{F} , private pattern set $\mathcal{P}_s (\mathcal{P}_s \subset \mathcal{F})$

Output: privacy-free frequent pattern set \mathcal{F}'

Method:

```

1:  $\mathcal{F}' = \mathcal{F};$ 
2: while  $\mathcal{P}_s \neq \emptyset$  do begin
3:    $S_c = \emptyset;$ 
4:   for each private pattern  $q \in \mathcal{P}_s$  do
5:     for each item  $i \in q$  do
6:       if  $i \notin S_c$  then  $S_c = S_c \cup \{i\}; fcnt[i] = 0; pcnt[i] = 1;$ 
7:       else  $pcnt[i] = pcnt[i] + 1;$ 
8:     for each  $p \in \mathcal{F}'$  do  $\forall i \in (p \cap S_c), fcnt[i] = fcnt[i] + 1;$ 
9:     for each  $i \in S_c$  do  $s[i] = pcnt[i]/fcnt[i];$ 
10:    select item  $x \in S_c$  such that  $x = \arg \max_{i \in S_c} \{s[i]\};$ 
11:     $\mathcal{F}' = \mathcal{F}' - \{p | (x \in p) \wedge (p \in \mathcal{F}')\};$ 
12:     $\mathcal{P}_s = \mathcal{P}_s - \{q | (x \in q) \wedge (q \in \mathcal{P}_s)\};$ 
13: end

```

Fig. 3. Algorithm MAXSIPS

number of iterations in pattern sanitization, and finally maximize the number of non-private patterns in \mathcal{F}' . Similarly, according to Theorem 2, we can show that MAXSIPS generates a privacy-free frequent pattern set.

5 Experiments

In this section, we evaluate the performance of our algorithms RANDIPS, MINCIPS and MAXSIPS. All these algorithms are implemented with C++.

5.1 Experimental Settings

We use a synthetic dataset, T40I10D100K, generated by IBM data generator. The details of the generation procedure and related parameters are described in [13]. In our experiments, we set $N = 1000$, $|L| = 2000$, $|T| = 40$, $|I| = 10$, $|D| = 100K$. There are 1000 different items, and 100K transactions in our test dataset. Each transaction contains 40 items on average.

The methodology of our experiments is as follows. Given minimum support threshold σ , we first get the set \mathcal{F} of all frequent patterns by mining the test dataset with Apriori [13]. Then, we randomly choose some patterns from \mathcal{F} as the set \mathcal{P}_s of private patterns such that $\forall q_1, q_2 \in \mathcal{P}_s, q_1 \not\subseteq q_2$ and $q_2 \not\subseteq q_1$. Because if $q_1 \subseteq q_2$, hiding q_1 will also protect q_2 from disclosure simultaneously. Similarly, when $q_2 \subseteq q_1$, we only need considering to hide q_2 . We generate privacy-free frequent pattern set \mathcal{F}' by sanitizing \mathcal{F} with one of the algorithms RANDIPS, MINCIPS, and MAXSIPS. Because the content of private patterns may have impact on the result of pattern sanitization, we plot each result in the following figures with an average over the results of 10 trials.

All our experiments are performed on a 1.5GHz Intel Pentium IV PC with 256MB main memory, running Microsoft Window XP operating system.

5.2 Effectiveness

We compare the effectiveness of RANDIPS, MINCIPS and MAXSIPS in this set of experiments. The effectiveness is measured with the size of the resultant privacy-free frequent pattern set. In Fig. 4(a), we report the experimental results with respect to the number of private patterns. In our experiments, the set \mathcal{F} of frequent patterns is obtained by mining test dataset with minimum support threshold $\sigma = 1.5\%$. We randomly choose 10, 20, 30, 40, 50 patterns from \mathcal{F} to form five sets of private patterns. It can be seen that MAXSIPS is more effective than both RANDIPS and MINCIPS, irrespective of the number of private patterns. The reason is that MAXSIPS takes into consideration both frequent patterns and private patterns when choosing victim for pattern sanitization. However, MINCIPS only emphasizes considering frequent patterns, and RANDIPS selects victim randomly. In Fig. 4(a), we can also see that with the number of private patterns increasing, more frequent patterns need to be eliminated for preventing the disclosure of private patterns. Hence, the size of the privacy-free frequent pattern set decreases for all algorithms.

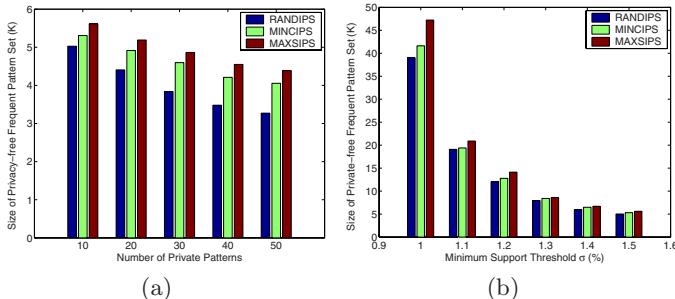


Fig. 4. Effectiveness of Algorithms

We also compare the effectiveness of algorithms by varying the minimum support threshold σ for frequent pattern mining. The results are shown in Fig. 4(b). The value of σ ranges from 1% to 1.5% in our experiments. The smaller value of σ is, the more number of frequent patterns will be mined. Each time when we obtain a set \mathcal{F} of frequent patterns, we randomly choose 10 patterns from \mathcal{F} to form the set \mathcal{P}_s of private patterns. It can be seen that MAXSIPS can obtain a privacy-free frequent pattern set with larger size than both RANDIPS and MINCIPS under all the settings of minimum support threshold σ . Especially when given a smaller σ , the performance of MAXSIPS is much better than that of RANDIPS and MINCIPS.

5.3 Time for Pattern Sanitization

In this set of experiments, we compare the execution time of RANDIPS, MINCIPS and MAXSIPS. Fig. 5(a) shows their execution time with respect to the number of private patterns. In our experiments, the set \mathcal{F} of frequent patterns is obtained with minimum support threshold $\sigma = 1.5\%$. We increase the number of private patterns from 10 to 50. For each private pattern, RANDIPS only scans the set of frequent patterns at most once for pattern sanitization, while MINCIPS and MAXSIPS may scan twice the set of frequent patterns, one for calculating the counter f_{cnt} , and another for pattern sanitization. Hence, RANDIPS spends less time on pattern sanitization than both MINCIPS and MAXSIPS. An interesting result in Fig. 5(a) is that the execution time of MAXSIPS is less than that of MINCIPS when the number of private patterns is larger than 30. The reason is that MAXSIPS can hide more private patterns in an iteration of pattern sanitization than MINCIPS does. Thus, MAXSIPS will scan the set of frequent patterns in less iterations than MINCIPS.

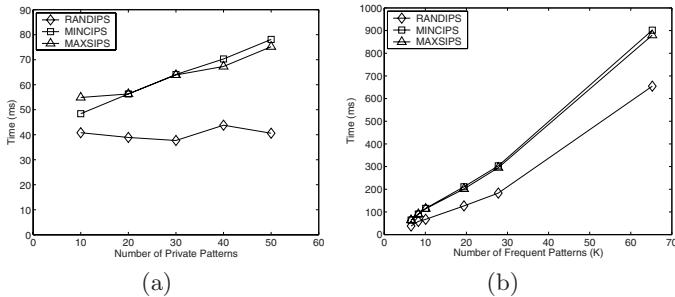


Fig. 5. Execution Time of Algorithms

In Fig. 5(b), we compare the execution time of algorithms with respect to the number of frequent patterns. By varying minimum support threshold σ from 1.5% to 1%, we obtain five sets of frequent patterns, where the number of frequent patterns ranges from 6539 to 65236. For each set of frequent patterns, we randomly choose 30 patterns from it as the set of private patterns. Since MINCIPS and MAXSIPS scan more times the set of frequent patterns than RANDIPS does, the increases in their execution time are more than that of RANDIPS when the number of frequent patterns increases. Fig. 5(b) also shows the interesting result similar to that in Fig. 5(a). That is, when we increase the number of frequent patterns, MAXSIPS spends less time on pattern sanitization than MINCIPS. The reason is that MAXSIPS maximizes the number of private patterns being hidden in each iteration of pattern sanitization. As such, it needs less iterations during pattern sanitization. Hence, when increasing the number of frequent patterns, the execution time of MAXSIPS becomes less than that of MINCIPS.

5.4 Scalability

In this set of experiments, we test on the scalability of algorithms MINCIPS and MAXSIPS. We show the scalability of MINCIPS and MAXSIPS with respect to the number of private patterns in Fig. 6(a) and Fig. 6(b), respectively. In our experiment, we increase the number of private patterns from 10 to 50, and repeat the same experiment with different settings of minimum support threshold σ . It can be seen that the execution time of MINCIPS and MAXSIPS always increases linearly with respect to the number of private patterns.

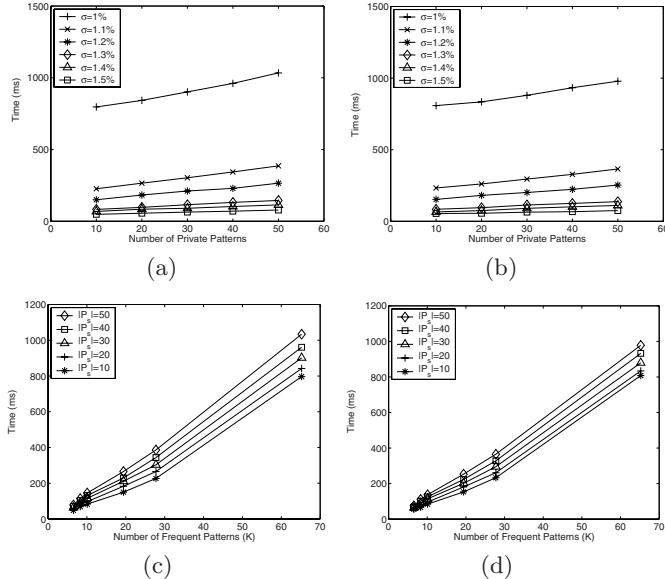


Fig. 6. Scalability

In Fig. 6(c) and Fig. 6(d), we show the scalability of MINCIPS and MAXSIPS with respect to the number of frequent patterns, respectively. By mining test dataset with different minimum support threshold σ , we can vary the number of frequent patterns in \mathcal{F} . In our experiment, we decrease σ from 1.5% to 1%, and the number of discovered frequent patterns increases from 6539 to 65236. The same experiment is also repeated on the set \mathcal{P}_s with different number of private patterns. We can see that for both MINCIPS and MAXSIPS, the execution time also increases linearly with respect to the number of frequent patterns.

6 Conclusions

Sharing the knowledge discovered by data mining without discrimination may cause the disclosure of sensitive information. In this paper, we have addressed the problem of privacy-preserving frequent pattern sharing. We present the notation

of privacy-free frequent pattern set, and show that finding an optimal solution to the problem of privacy-preserving frequent pattern sharing is NP-hard. Then, we propose a framework of item-based pattern sanitization, and present three heuristic algorithms for privacy-preserving frequent pattern sharing. Our experimental results show that MAXSIPS is more effective amongst them, and its execution time is linearly scalable with respect to both the number of frequent patterns and the number of private patterns.

For future research, we will investigate the possibility of developing more effective and efficient algorithms for privacy-preserving frequent pattern sharing, and we will also extend our research to other forms of knowledge representation.

References

1. Calders, T.: Computational complexity of itemset frequency satisfiability. In: PODS, Paris, France (2004) 143–154
2. T.Mielikainen: On inverse frequent set mining. In: Workshop on Privacy Preserving Data Mining. (2003) 18–23
3. Clifton, C., Marks, D.: Security and privacy implications of data mining. In: ACM SIGMOD Workshop on Data Mining and Knowledge Discovery, Montreal, Canada (1996) 15–19
4. Han, J., Kamber, M.: Data Mining: Concept and Techniques. Morgan Kaufmann Publishers, San Francisco (2000)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
6. Oliveira, S.R.M., Zaïane, O.R., Saygin, Y.: Secure association rule sharing. In: PAKDD, Sydney, Australia (2004) 74–85
7. Wang, Z., Wang, W., Shi, B., Boey, S.: Preserving private knowledge in frequent pattern mining. In: IEEE ICDM Workshop on Privacy Aspects of Data Mining, Hong Kong, China (2006) 530–534
8. Atzori, M., Bonchi, F., Giannotti, F., Pedreschi, D.: k -anonymous patterns. In: PKDD, Porto, Portugal (2005) 10–21
9. Atallah, M., Bertino, E., Elmagarmid, A., Ibrahim, M., Verykios, V.: Disclosure limitation of sensitive rules. In: IEEE Knowledge and Data Engineering Exchange Workshop, Chicago, IL (1999) 45–52
10. Oliveira, S.R.M., Zaïane, O.R.: Algorithms for balancing privacy and knowledge discovery in association rule mining. In: IDEAS, Hong Kong, China (2003) 54–65
11. Saygin, Y., Verykios, V.S., Clifton, C.: Using unknowns to prevent discovery of association rules. SIGMOD Record **30**(4) (2001) 45–54
12. Wang, Z., Liu, B., Wang, W., Zhou, H., Shi, B.: An effective approach for hiding sensitive knowledge in data publishing. In: WAIM, Hong Kong, China (2006) 146–157
13. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB, Santiago, Chile (1994) 487–499

K_n Best - A Balanced Request Allocation Method for Distributed Information Systems*

Jorge-Arnulfo Quiané-Ruiz**, Philippe Lamarre, and Patrick Valduriez

INRIA and LINA

Université de Nantes

2 rue de la houssinière, 44322 Nantes Cedex 3, France

{Jorge.Quiane,Philippe.Lamarre}@univ-nantes.fr,Patrick.Valduriez@inria.fr

Abstract. In large-scale distributed information systems, providers are typically autonomous, i.e. free to leave the system at will or to perform certain requests. In this context, request allocation is critical for the efficient system's operation. However, most methods used in distributed information systems aim at maximizing overall system performance (throughput and response times) by allocating requests to the most efficient providers, without considering providers' autonomy. In this paper, we propose a balanced request allocation method, K_n Best, which considers providers' autonomy in addition to load balancing. Our method is general and simple, so that it can be easily incorporated in existing distributed information systems. We describe the implementation of K_n Best in different scenarios. Finally, we give an experimental evaluation which shows that K_n Best significantly outperforms traditional request allocation methods.

1 Introduction

We consider distributed information systems that are dynamic and provide access to a large number of information providers. Providers can be fairly autonomous, i.e. free to leave the system at will and to express their *intentions* for performing requests. Their *intentions* can stem from combining their *preferences* with other important factors such as their *strategies*.

In this context, allocating requests to providers must maximize overall system performance (throughput and response times). The traditional solution, used in mediator systems [6][14][17], is to reduce the *overloading* of providers as much as possible so that load balancing is increased. However, preserving the providers' *intentions* when performing request allocation is equally important to keep providers happy and the system stable. With the traditional solution, providers can become unsatisfied when their intentions are not met and simply quit, resulting in an unstable system. Therefore, request allocation should

* Work partially funded by ARA “Massive Data” of the French ministry of research (projects MDP2P and Respire) and the European Strep Grid4All project.

** This author is supported by the Mexican National Council for Science and Technology (CONACyT).

also take into account providers' *intentions*. This is a timely problem with the deployment of web services and service-oriented architectures.

Providers' *intentions* are dynamic and depend on the providers' context. For instance, some providers may desire to perform specific requests at some time, but not at another time. Furthermore, providers can be heterogeneous in terms of capacity and data. Heterogeneous capacity means that some providers are more powerful than others and can treat more requests per time unit. Data heterogeneity means that different providers provide different data and thus produce different results for the same request.

In this paper, we address the request allocation problem by considering providers' *intentions* in addition to load balancing. We propose a balanced request allocation method, called K_n *Best*. It is inspired by the *two random choices* paradigm which has proven useful for dynamically assigning tasks to providers [10,20,21]. Our method is general and simple, so that it can be easily incorporated in existing distributed information systems. It generalizes traditional methods and can be adapted to the application by varying several parameters. We describe the implementation of K_n *Best* in different scenarios. Finally, we give an experimental evaluation which compares K_n *Best* to traditional request allocation methods. We show that, with autonomous information providers in the system, our method significantly outperforms these methods.

The rest of this paper is organized as follows. In Section 2, we define the system model and introduce some notations. In Section 3, we present the K_n *Best* method. In Section 4, we give the experimental evaluation of K_n *Best*. In Section 5, we discuss related works. Section 6 concludes.

2 Model and Notations

We consider a system consisting of a set of mediators, M , of a set of consumers, C , and of a set of providers, P . These sets are not necessarily disjoint, i.e. we can have $M \cap C \cap P \neq \emptyset$. Requests are formulated in a format abstracted as a triple $q = < c, d, n >$ such that $q.c \in C$ is the identifier of the consumer that has issued the request, $q.d$ is the description of the task to be done, and $q.n \in \mathbb{N}^*$ is the number of providers to which the consumer wishes to allocate its request. Consumers send their requests to a mediator $m \in M$ which allocates each incoming request q to the $q.n$ providers in P_q , where P_q denotes the set of providers that can treat q . We use N_q for denoting $|P_q|$, or simply N for the sake of simplicity when there is no ambiguity on q . In the case where $q.n > N_q$, the consumer $q.c$ will get N_q results instead of $q.n$. We only consider the arrival of feasible requests, that is, those requests for which $P_q \neq \emptyset$.

Request allocation of some feasible request q among the providers which are able to deal with q is defined as a vector $\overrightarrow{\text{Alloc}}$ or $\overrightarrow{\text{Alloc}}_q$ when there is an ambiguity on q (see Definition 1). The set of providers such that $\overrightarrow{\text{Alloc}}[p] = 1$ is noted \widehat{P}_q .

Definition 1. Request Allocation

The allocation of request q among the providers in P_q is a vector \vec{Alloc} of length N such that: $\forall p \in P_q$, $\vec{Alloc}[p] = \begin{cases} 1 & \text{if provider } p \text{ gets the request} \\ 0 & \text{otherwise} \end{cases}$

$$\text{with } \sum_{p \in P_q} \vec{Alloc}[p] = \min(q.n, N)$$

Each provider $p \in P$ has a finite *capacity*, $cap(p) > 0$, for performing feasible requests. The *capacity* of a provider denotes the number of computational units that it can have. Similarly, each feasible request q has a *cost*, $cost_p(q) > 0$, that represents the computational units that q consumes at p . We define provider *utilization* as follows.

Definition 2. Provider Utilization

Let Q_p denote the set of requests that have been allocated to p but have not already been treated at time t (i.e. the pending requests at p). The utilization of a given provider $p \in P$ at time t , $U_t(p)$, is defined as the computational units that Q_p consumes at p

$$U_t(p) = \frac{\sum_{q \in Q_p} cost_p(q)}{cap(p)}$$

Providers are free to express their *intentions* for performing each feasible request q , denoted by the $PI_p(q)$ function whose values are between $-\infty$ and 1, and where p denotes a given provider. If the *intention* value is positive, the greater it is, the greater the desire for performing requests. If the *intention* is negative, the smaller it is, the greater the refusal for performing requests. Providers' refusal can go down to $-\infty$ because *utilization* can, theoretically, grow up to $+\infty$. In order to guarantee system stability, the way in which such *intentions* are computed is considered as private information for providers.

3 K_nBest Method

In this section, we present the *K_nBest* method for balanced request allocation. It is meant to be run by one or several mediators which allocate the requests they receive from the clients. We restrict ourselves to the case where requests can be viewed as single units of work called *tasks*. An incoming feasible request q can be allocated to n providers (because of data heterogeneity, more than one provider can provide answers). We assume that there is a matchmaking mechanism (§ for example) to find the set P_q of providers that are able to deal with each incoming feasible request q . Therefore, we can focus on request allocation only. Our method is inspired by the *two random choices* paradigm [10,20,21]. The principle of *two random choices* is to randomly select a set of providers K among the N_q providers and then allocate the request to the least utilized provider in K . *K_nBest* uses a similar principle. We describe below the principle and properties of the *K_nBest*.

3.1 K_n Best Principle

Given an incoming feasible request q and the set P_q of providers which are able to deal with it, we denote the providers' *intention* for dealing with q by the vector \vec{PI}^q . Algorithm 1 shows the main steps of K_n Best for allocating q .

First (line 2 of Algorithm 1), a set K of providers are selected at random among the set P_q , where $|K| = k$ and $k \in \mathbb{N}^*$. Second (line 3), the $k_n \in \mathbb{N}^*$ least utilized providers (i.e. the K_n set) are selected among the set K of providers¹. Third (line 4), the providers' *intention* vector, $\vec{PI}^q[p]$, for dealing with q is computed. Considering our system architecture [15], this operation is realized at the mediator sites and do not impact the network traffic. Next (line 5), the *intentions'* ranking vector \vec{R}^q is computed. This ranking is necessary for the selection of providers to deal with q . Intuitively, $\vec{R}^q[1]$ is the most interested provider to deal with q , $\vec{R}^q[2]$ the second, and so on until to $\vec{R}^q[N]$ which is the least interested. Finally (lines 6-8), the request q is allocated to the n best providers. If $N < n$, the request is simply allocated to all N providers without any further considerations. The second (line 3) and fourth (line 5) phases can be solved using a sorting algorithm. So, in the worst case, their complexity is $O(k \log_2(k))$ and $O(k_n \log_2(k_n))$, respectively.

Algorithm 1: K_n Best Providers

```

Input :  $q, k, k_n, P_q$ 
Output:  $\vec{Alloc}_q$ 
1 begin
2    $K \leftarrow$  select, at random,  $k$  providers in the set  $P_q$  ;
3    $K_n \leftarrow$  select the  $k_n$  less utilized providers in the set  $K$  ;
4   foreach  $p \in K_n$  do fork ask for the provider's intention  $\vec{PI}^q[p]$  ;
5   compute the providers' intention vector ranking  $\vec{R}^q$  ;
6   for  $i = 1$  to  $\min(n, k_n)$  do  $\vec{Alloc}_q[\vec{R}^q[i]] \leftarrow 1$  ;
7   for  $j = \min(n, k_n) + 1$  to  $k_n$  do  $\vec{Alloc}_q[\vec{R}^q[j]] \leftarrow 0$  ;
8 end
```

3.2 K_n Best Property and Analysis

An expected property of providers is *individual rationality*, whereby they behave according to their own interests only [1]. However, this may lead providers to not participate in a given request allocation since they are *selfish* and may have no interest in processing the request. Thus, *individual rationality* is incompatible with the system's objectives to process efficiently all incoming feasible requests and satisfy consumers. For these reasons, we are looking for a special kind of *long-term individual rationality* which K_n Best allows.

¹ We can indifferently assume that the values of k and k_n are predefined by the administrator or defined on the fly by mediators.

In the case of m requests and m homogeneous providers, request allocation methods based on the *two random choices* paradigm (which we call the *TRCBased* methods) assign to providers a maximum load of only $\theta(\log \log m)$ with high probability [10][20]. Given an incoming request q , *TRCBased* methods randomly select two providers (the K set) among P_q and allocate q to $p \in K$, such that

$$\forall p' \in K \setminus \widehat{P}_q, U_t(p') \geq U_t(p)$$

Nonetheless, the *TRCBased* methods are only suitable for homogeneous distributed systems (i.e. providers have the same *capacity* for performing requests). In the context of heterogeneous distributed systems, request allocation methods ensure good performances in the system by considering the capacities of all the providers in P_q [6][12][14][17][21]. These methods (the *CapacityBased* methods) allocate each incoming request q to $p \in P_q$ such that

$$\#\{p' \in P_q \setminus \widehat{P}_q : U_t(p') < U_t(p)\}$$

A third possible way to allocate requests is by only considering the providers' *intentions*, which we call the *IntentionBased* methods. Given the request q to be allocated, *IntentionBased* methods allocate q to $p \in P_q$ such that

$$\#\{p' \in P_q \setminus \widehat{P}_q : PI_{p'}(q) \geq PI_p(q)\}$$

On the other hand, the k_n parameter defines the general behavior of the *K_nBest* method. For smaller values of k_n , *K_nBest* allocates the requests by mainly considering the providers' *utilization*, and by basically considering the providers' *intentions* for greater values of k_n . Thus, the value that k_n may take depends on the type of application. Therefore, we recommend smaller values of k_n if the system pays more attention to the providers' *utilization* and greater values if it pays more attention to the providers' *intention*. In the following theorem, we summarize the *K_nBest* properties which bound its behavior.

Theorem 1. *For a large number of heterogeneous providers, K_nBest behavior is bounded by the following properties.*

- (i) if $k = 2q.n \wedge k_n = q.n$, *K_nBest* is equivalent to the *TRCBased* methods.
- (ii) if $k = N_q \wedge k_n = q.n$, *K_nBest* is equivalent to the *CapacityBased* methods.
- (iii) if $k = N_q \wedge k_n = k$, *K_nBest* is equivalent to the *IntentionBased* methods.

Proof. Omitted because of triviality.

A different way to proceed in *K_nBest* is by selecting first the most *interested* providers (K_n) among K to finally allocate requests to the least *utilized* providers among K_n . If request allocation is done this way, *intentions* are given more importance than *utilization*. Besides, one can replace the providers' *intentions* by the consumers' *intentions* for allocating requests if the objective of the system is, for example, to ensure interesting results for consumers. If the system's goal is to satisfy both providers and consumers, the consumers' *intentions* can be considered as an additional step in *K_nBest*, or *intentions* of both providers and consumers can be merged in only one step.

4 Experimental Evaluation

In this section, we give an experimental evaluation of K_n Best using simulation. We carried out several series of tests with a main objective in mind: *how well K_n Best operates in environments with heterogeneous and autonomous providers*. We assume a single mediator. In order to assess the quality of K_n Best, we conducted three types of evaluations: *performance*, *request load balance*, and *satisfaction balance*. In the following, we introduce the baseline methods to which we compare K_n Best and our experimental setup, and then we present the experimental results.

4.1 Baseline Methods

In distributed information systems, the well-known methods to allocate requests efficiently across providers are those with the *CapacityBased* behaviour (i.e. the *CapacityBased* methods) [6][4][7]. These methods, unlike the *TRCBased* ones, operate well in heterogeneous systems which is why we use them as baseline.

Economic models have been shown to provide efficient request allocation in heterogeneous systems [3][4][6][13]. We use, as baseline, an *Economic* method whose criterion to select providers is the *utilization* and *bid* of providers. Note that different economic methods may have other performance results than those presented here. In our experiments, we use virtual money (*token*) which is just seen as a means of regulation. In the course of time, the *tokens* are spent by providers in order to acquire requests. Thus, a source of financing is necessary to them. Otherwise, after some time, providers would not have more *tokens* to bid positively. We have chosen to associate a bank with the mediator, which gives a specific amount of *tokens* to providers at the registration step and equally redistributes the *tokens* in the course of time.

4.2 Experiment Setup

We built a Java-based simulator that models a *mono-mediator* distributed system, following the system architecture in [15]. In all experiments, the number of consumers and providers is 200 and 400 respectively, with only one mediator allocating all the incoming feasible requests. We assign sufficient resources to the mediator so that it does not cause bottlenecks in the system. Each result we present is obtained by 10 simulation series. Feasible requests arrive to the system following the *Poisson* distribution, which has been found in dynamic and heterogeneous distributed systems [5].

For our simulations, we consider that providers provide answers with similar quality, hence assuming that consumers always ask for 1 provider to solve their requests (i.e. $q.n = 1$). Since we focus on heterogeneous distributed systems, we set $k = N_q$. Basing ourselves on the results of [10][20], we set $k_n = 2q.n$ (i.e. $k_n = 2$). In order to compare the K_n providers, we consider their *intentions* for performing requests. Providers work out their *intentions* using the *S_bQLB* method [7]. To ensure high autonomy in our experiments, providers' *preferences*

Table 1. Simulation parameters

Parameter	Definition	Value
nbConsumers	Number of consumers	200
nbProviders	Number of providers	400
nbMediators	Number of mediators	1
qDistribution	Query arrival distribution	Poisson
iniSatisfaction	Initial providers' satisfaction	0
qClasses	Supported query classes	2
nbSimulations	Number of realized simulations for each experience test	10

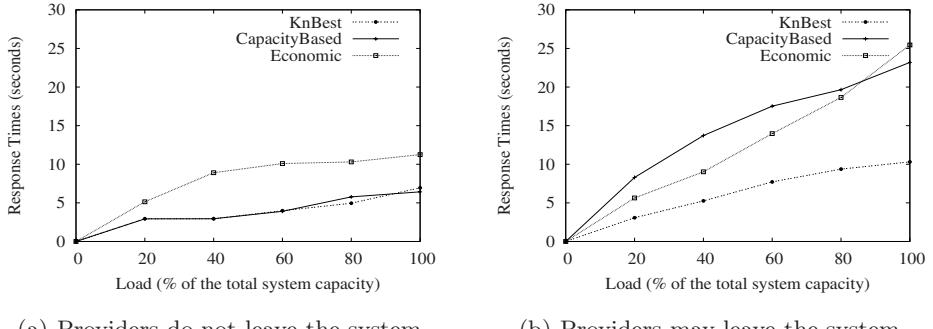
are randomly obtained between -1 and 1 . More sophisticated mechanisms for obtaining such preferences can be applied (2 for example).

Since our main focus in this paper is to study the way in which requests are allocated in the system, we do not take into account the bandwidth problem. It is because of this that simulation tests are enough for evaluating our method. Providers are initialized with a satisfaction value of 0 . We set the providers' capacity heterogeneity in accordance to the results in [18]. Based on these results, we generate around 10% of low-capable, 60% of medium-capable, and 30% of highly-capable providers. The highly-capable providers are 3 times more capable than medium-capable providers and still 6 times more capable than low-capable ones. We generate two classes of requests that consume, respectively, 130 and 150 computational units at the high-capable providers (taking about 1.3 and 1.5 seconds, respectively).

4.3 Experimental Results

Performance. We start with an evaluation of the system's response time guaranteed by the *CapacityBased*, *Economic*, and *K_nBest* methods. As is conventional, we define the response time as the elapsed time from the time a consumer issues the request to the time it receives the answer. As first case, we consider that providers are *captive* and do not leave the system by *unsatisfaction*, *request starvation*, nor *overutilization*. Figure 1(a) shows that *K_nBest* yields the same response times as *CapacityBased*, and that both of them outperform the *Economic* method. This shows that *K_nBest* is also suitable for environments where providers do not leave the system.

As second case, in order to evaluate the impact of provider departures, we study response time when providers may leave the system by *unsatisfaction* or *request starvation*. To do so, we allow providers to leave the system if their *satisfaction* is 0.15 under their *adequation*. The *adequation* notion (see 7 for details) denotes the degree of satisfaction towards all the feasible requests proposed by the system. Also, providers are allowed to leave the system if they do not perform at least 25% of what they should. The results are shown in Figure 1(b). We observe that *K_nBest* significantly outperforms the *CapacityBased* and *Economic* methods. While *K_nBest* degrades in average the system's response time by a factor of 1.5 only, the *Economic* and *CapacityBased* do it, respectively,

**Fig. 1.** Response Time

by a factor of 1.8 and 4! We observed that in the *Economic* method, providers usually quit by *request starvation*, while in the *CapacityBased*, they do so by *unsatisfaction*.

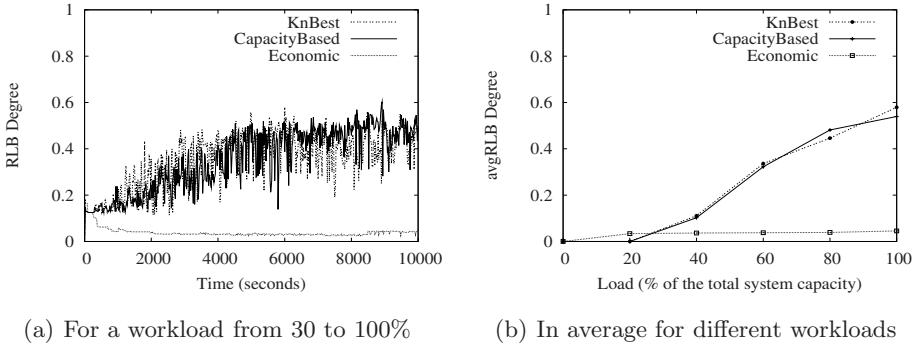
We ran other experiments where providers may also leave the system by *overutilization*², but by lack of space, we do not present the results here. We observed that *KnBest* and *CapacityBased* do not suffer from providers' departures by *overutilization*, while the performance of the *Economic* method is degraded by a factor of 4.5 and, worse again, at a given time the system remains with no providers! These results demonstrate the great impact on response times of providers departures by *unsatisfaction*, *request starvation*, and *overutilization*. Therefore, in a system where providers are *selfish*, we must pay special attention to their interests.

Request Load Balance. We now study the *request load balance* (*RLB*) for various workloads. In the experiments, we measure the *RLB* at any time as the ratio of the smallest and largest *utilized* providers. Furthermore, it is important that the system strives to give requests to all providers in the system if possible so they don't leave. Thus, we also measure the *average request load balance* (*avgRLB*) at a discrete time interval $[t_1, t_2]$. The *avgRLB* measure is symmetrical to the *RLB* measure. The *average utilization*, $U_{[t_1, t_2]}(p)$, is said to be the average *utilization* of the provider p at the discrete time interval $[t_1, t_2]$.

We know that the thresholds of *request starvation* and *overutilization* over which providers decide to leave, are very subjective and might depend on several external factors. Thus, to avoid any question on the choice of such *thresholds*, we assume, in this study, that providers are not allowed to leave the system whatever their degree of *request starvation* and *overutilization* are.

Contrary to the expected, on the one hand, the results show that the *E-economic* method has serious problems to ensure good *RLB* ratios in the system, despite that providers take more into account their *preferences* than their

² For instance, when providers want to guarantee a good *quality of service*.

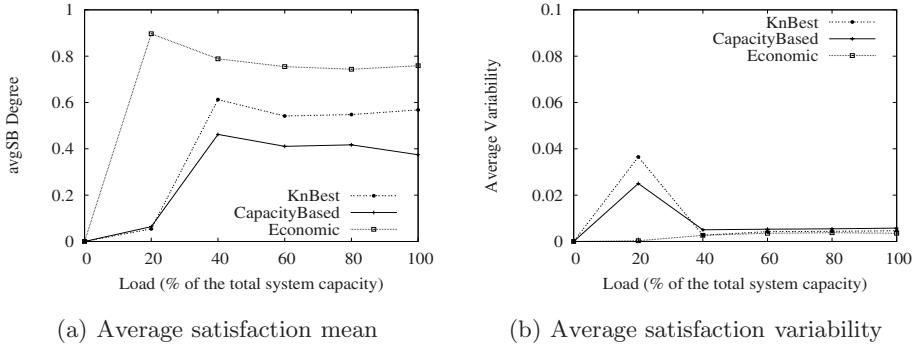
**Fig. 2.** Request load balance

utilization at the time of bidding for requests. On the other hand, the results show that *CapacityBased* and *K_nBest* have problems to ensure good *RLB* only for workloads under the 40% of the total system capacity. When workloads increase, both algorithms improve the *RLB* in the system. This is because most providers in the first case have almost no requests (i.e. have all their capacity available), and thus, requests may be allocated to those providers that spend more treatment units to perform them. Hence, each time that requests are allocated to the least capable providers, the distance between the most and least utilized providers increases significantly. For lack of space, we do not present these results.

In order to explain the above phenomenon, we present the results of a series of simulations where we uniformly vary the workload from 30% (at the beginning of the simulation) to 100% (at the end of the simulation). The results show that *CapacityBased* and *K_nBest* methods effectively improve the *RLB* as the workload increases, and that the *Economic* method cannot ensure an acceptable *RLB* in the system (see Figure 2(a)). In all cases the *K_nBest* performance is as good as *CapacityBased* one, even if the former takes into account the providers' intentions and the latter does not.

Let us analyze the *avgRLB* guaranteed by these three methods, that is, how well these methods avoid *request starvation*. To this end, we measure the *avgRLB* in a time interval of 20 seconds over a simulation of 10000 seconds for different workloads. The results are shown in Figure 2(b). We observe that, unlike the *RLB* results, for workloads over 20% of the total system capacity the *CapacityBased* and *K_nBest* methods significantly outperform the *Economic* method. This means that while *CapacityBased* and *K_nBest* strive to allocate requests on giving requests to all the providers in the system, the *Economic* method suffers from serious *request starvation problems*.

Satisfaction Balance. We now study the *satisfaction balance (SB)* ratios guaranteed by the three methods for various workloads. We assume that providers work out their *satisfaction* as in [7], but without loss of generality, providers

**Fig. 3.** Providers' satisfaction in average

can do it differently. For lack of space, we only present the *average satisfaction balance* (*avgSB*) measures, at a discrete time interval $[t_1, t_2]$. The *avgSB* is defined as the ratio of the smallest and largest *average satisfied* providers, where *average satisfaction* denotes the average of the providers' *satisfaction* at the discrete time interval $[t_1, t_2]$. Furthermore, we measure the *standard deviation* of the providers' *average satisfaction* (*average variability*), in order to evaluate how these methods satisfy all providers. Similarly to the thresholds of *request starvation* and *overutilization*, the *unsatisfaction* threshold over which providers decide to leave, is also quite subjective and may depend on several external factors. Then, to avoid any question on the choice of such a threshold, we assume that providers are not allowed to leave the system by *unsatisfaction*.

Conversely to the *RLB* results and as expected (because the providers' bids are based on the providers' *intentions*), the *Economic* method preserves better the providers' *intentions* (see Figure 3(a)). But this occurs because *Economic* does not pay much attention to the providers' *utilization*. Hence, this method is only suitable for systems where providers do not care about their *utilization* nor response time is very important for consumers. In these results, we have also observed that even if K_n *Best* ensures the same *RLB* than *CapacityBased*, it significantly satisfies better providers. Nevertheless, we can observe that both methods have some difficulties to preserve the providers' *intentions* for a very low workload (20% the total system capacity). This is due to the fact that the number of incoming feasible requests is not enough for giving requests to all providers, and thus in both methods, requests are allocated to the least *utilized* providers. Note that K_n *Best* can be improved in two ways: 1) setting the k_n to a greater value, or 2) in selecting first the k_n most interested providers and allocating requests to the $q.n$ least utilized ones in K_n .

Figure 3(b) shows that, for workloads under 20% of the total system capacity, some providers get more satisfied than others in the *CapacityBased* and K_n *Best* methods. This is why the *average variability* of both methods is high. However, for higher workloads, the three methods yield almost the same *average variability*.

5 Related Work

In the context of large-scale distributed information systems, most of the work on request allocation [6,14,17,20,21] has dealt with the problem of allocating requests to the most capable providers. However, preserving providers' *autonomy* for performing requests has not received much attention. Economical methods [3,4,16,13] strive to deal with such *autonomy*. In such models, every provider tries to maximize its revenue by selling services to consumers. Mariposa [13] pioneered the use of an economical method for dealing with the request allocation problem in distributed systems. In Mariposa, all the incoming requests are processed by a *broker site*, which given an incoming request, requests providers for bids. Providers bid for acquiring the request based on a local bulletin board. Then, the *broker site* selects a set of bids that has an aggregate price and delay under a bid curve provided by the consumer. Nevertheless, it is unclear if this method ensures good response times in adequacy with the providers' *intentions*. Furthermore, some requests may not get processed although it exist providers able to deal with it. Recently we have proposed the *S_bQLB* approach [7] to balance requests across providers while considering their *intentions* for performing requests. *S_bQLB* strives to balance requests in the course of time thereby reducing *request starvation* in the system. However, *K_nBest* is orthogonal and more general than all these methods. The *K_nBest* method is actually a set of solutions for different environments and types of applications.

6 Conclusion

In this paper, we addressed the problem of request allocation in large-scale distributed information systems with autonomous providers. We proposed the *K_nBest* method which allocates requests by considering providers' *intentions* for performing requests in addition to their *utilization*. The principle of *K_nBest* is similar to the *two random choices* paradigm which has proven useful for dynamically assigning tasks to providers. We described the implementation of *K_nBest* in different scenarios with the different strategies to adopt. We gave an experimental evaluation which compares *K_nBest* to traditional request allocation methods. We demonstrated through experimentation that *CapacityBased* and *Economic* are not suitable for distributed information systems where providers may leave by *request starvation*, *overutilization*, or *unsatisfaction*. The experimental results show that, with autonomous providers in the system, our method significantly outperforms these methods. *K_nBest* is suitable for dynamic, very large-scale distributed information systems with competitive or cooperative behaviors. Our method is general and simple, so that it can be easily incorporated in existing distributed information systems. It generalizes traditional methods and can be adapted to the application by varying several parameters.

References

1. O. Morgenstern and J. von Neumann: *Theory of Games and Economic Behavior*. Princeton University Press, Inc. 1980.
2. A. Sah, J. Blow, and B. Dennis: An introduction to the Rush language. In Procs. of the TCL Workshop. 1994.
3. D. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini: Economic Models for Allocating Resources in Computer Systems. *Market-based control: a paradigm for distributed resource allocation*. World Scientific Publishing Co., Inc. 1996.
4. D. Ferguson, Y. Yemini, and C. Nikolaou: Microeconomic Algorithms for Load Balancing in Distributed Computer Systems. In Procs. of the ICDCS Conference. 1988.
5. E. P. Markatos: Tracing a Large-Scale Peer to Peer System: An Hour in the Life of Gnutella. In CCGRID Symposium. 2002.
6. H. Zhu, T. Yang, Q. Zheng, D. Watson, O. Ibarra, and T. Smith: Adaptive Load Sharing for Clustered Digital Library Servers. In HPDC Symposium. 1998.
7. J.-A. Quiané-Ruiz, P. Lamarre, and P. Valduriez: Satisfaction-Based Query Load Balancing. In Procs. of the CoopIS Conference. 2006.
8. K. Sycara, M. Klusch, S. Widoff, and J. Lu: Dynamic Service Matchmaking Among Agents in Open Information Environments. In SIGMOD Record 28(1). 1999.
9. L. Li and I. Horrocks: A Software Framework for Matchmaking Based on Semantic Web Technology. In Procs. of the WWW Conference. 2003.
10. M. Mitzenmacher: The Power of Two Choices in Randomized Load Balancing. PhD. Thesis, UC Berkeley, 1996
11. M. Nodine, W. Bohrer, and A. Ngu: Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth. In Procs. of the ICDE Conference. 1999.
12. M. Roussopoulos and M. Baker: Practical Load Balancing for Content Requests in Peer-to-Peer Networks. Distributed Computing 18(6):421-434. 2006.
13. M. Stonebraker, P. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu: Mariposa: A Wide-Area Distributed Database System. In VLDB J. 5(1). 1996.
14. N. Shivaratri, P. Krueger, and M. Singhal: Load Distributing for Locally Distributed Systems. In IEEE Computer Journal 25(12). 1992
15. P. Lamarre, S. Cazalens, S. Lemp, and P. Valduriez: A Flexible Mediation Process for Large Distributed Information Systems. In Procs. of the CoopIS Conference. 2004.
16. R. Buyya, H. Stockinger, J. Giddy, and D. Abramson: Economic Models for Management of Resources in Grid Computing. In CoRR Journal. 2001.
17. R. Mirchandaney, D. Towsley, and J. Stankovic: Adaptive Load Sharing in Heterogeneous Distributed Systems. In Parallel and Distributed Computing J. 9(4). 1990
18. S. Saroiu, P. Krishna Gummadi, and S. Gribble: A Measurement Study of Peer-to-Peer File Sharing Systems. In Procs. of the MCN Conference. 2002.
19. T. Özsü and P. Valduriez: *Principles of Distributed Database Systems*, (2nd ed.). Prentice-Hall, Inc. 1999.
20. Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal: Balanced Allocations. In SIAM Journal on Computing 29(1). 1999.
21. Z. Genova and K. Christensen: Challenges in URL Switching for Implementing Globally Distributed Web Sites. In Procs. of the ICPP Workshops. 2000.

The Circular Two-Phase Commit Protocol

Heine Kolltveit and Svein-Olaf Hvasshovd

Department of Computer and Information Science
Norwegian University of Science and Technology

Abstract. Distributed transactional systems require an atomic commitment protocol to preserve atomicity of the ACID properties. However, the industry leading standard, 2PC, is slow and adds a significant overhead to transaction processing. In this paper, a new atomic commitment protocol for main-memory primary-backup systems, C2PC, is proposed. It exploits replication to avoid disk-logging and performs the commit processing in a circular fashion. The analysis shows that C2PC has the same delay as 1PC, and reduces the total overhead compared to 2PC.

1 Introduction

Main memory prices have dropped significantly over the last years, and the state of many applications and databases can now be fitted entirely in main memory. To make the state both persistent and available, it can be replicated instead of written to disk. For instance, a backup replica (backup for brevity) takes over the processing if a primary replica (primary for brevity) fails. A backup is kept up to date by receiving the same operations as the primary (active replication [1]) or log records from the primary (passive replication [2]). The backup can either apply the log records to its own state or periodically receive checkpoints from the primary. Assuming that the *mean time to fail*, MTTF, is orders of magnitude larger than the *mean time to repair*, MTTR, the system only needs to be single fault tolerant to completely avoid the need for disk accesses. MTTR can be made very short by employing on-line self-repair mechanisms [3]. In addition, since disk accesses are slow compared to both RAM accesses and network latencies, replication can result in an improvement in performance.

A transaction is a collection of operations that transfers a system reliably from one state to another, while providing the ACID properties [4]: *Atomicity, consistency, isolation* and *durability*. Commonly, transaction termination and atomicity is satisfied by an *atomic commitment protocol*, ACP. The ACP has been showed to be an important factor of total transaction processing time and, in particular, the current industry leading standard, the Two-Phase Commit protocol, 2PC [5], is slow [6][7][8]. The delay caused by two rounds of messages and multiple log records flushed to disk cause a significant overhead. Also, a failure of the coordinator might block the participants from completing a transaction [9][10].

ACP performance and resilience to failures is a well established research field, but optimizations that will have significant effect is still possible under a parallel and replicated paradigm. Thus, this paper presents an ACP called Circular Two-Phase Commit

protocol, C2PC. It is an optimized version of 2PC for primary-backup systems. The protocol takes advantage of replication to trade costly flushed disk writes for cheaper message sends and RAM accesses. The idea is to send the vote and decision to the backup instead of a disk. This provides availability for the transaction participants and coordinator and renders 2PC *non-blocking* [9][10]. To give better performance, the vote and decision are sent in a ring instead of back and forth between the primary and backup. The protocol is always single fault-tolerant and these methods could be favorably applied in a shared-nothing, fault-tolerant DBMS like ClustRa [3].

The rest of the paper is organized as follows: Section 2 summarizes related work. Section 3 presents the system model and Section 4 defines the non-blocking atomic commitment problem. Section 5 gives an overview and a detailed description of C2PC, proves the correctness of the protocol and outlines a one-phased version called C1PC. Then, an evaluation of the protocols is given in Section 6. Finally, the conclusion and further work are presented in Section 7.

2 Related Work

Several atomic commitment protocols and variations have been proposed over the years. Many approaches have been concerned with either developing a non-blocking protocol or the performance issues. However, only a few deal with both.

In a non-replicated environment, 2PC may block if the coordinator and a participant fail [9][10]. 3PC [12] decreases the chance of blocking failures by adding an extra round of messages, thus favoring resilience over performance. 3PC has been extended to partitioned environments [13], and the number of communication steps has been reduced to the same as 2PC by using consensus [14], causing an increase in the number of messages or requiring broadcast capabilities.

Several 2PC-based modifications where performance issues are handled exist [15]. Presumed commit and presumed abort [16] both avoid one flushed disk write, by assuming that a non-existent log record means that the transaction has committed or aborted, respectively. Transfer-of-commit, lazy commit and read-only commit [9], sharing the log [16][17] and group commit [18][19] are other optimizations. An optimization of the presumed commit protocol [7] reduces the number of messages, but requires the same number of forced disk writes.

Optimistic commit protocols are designed to give better response time during normal processing, but will need extra recovery after failures or aborts. They release locks when the transaction is prepared, but must be able to handle cascading aborts by using semantic knowledge [20]. PROMPT [8] uses optimistic locking in the sense that locks can be lent to other transactions after the participant has voted yes. A transaction that lends locks will not reply to the request until the locks are fully released by the previous transaction, and only one transaction at a time can lend a lock. This approach avoids cascading aborts while it may yield better performance because of increased concurrency.

One-phased commit protocols have also been proposed [17][21][22][23][24]. These are based on the early prepare or unsolicited vote method by Stonebraker [25] where the prepare message is piggybacked on the last operation sent to a participant. In this way,

the voting phase is eliminated. However, these approaches inflicts strong assumptions and restrictions on the transactional system [22]. For instance, it requires either the participants to prepare the transaction for each request-reply interaction, or the coordinator must be able to identify the last request for a transaction to be able to piggyback a prepare-request. Otherwise, the performance of 1PC degrades.

A few approaches that render 2PC non-blocking by replication have been proposed. The first replicates the coordinator, but not the participants [11]. In addition to sending log records to the backup, they are forced to disk, causing a decrease in performance. Also, the backup only finishes transactions already started. No new transactions can be initiated by the backup. This approach has also been adapted to multiple backups [26].

The second combines optimistic commit and replication [27]. A replicated group of commit servers is used to keep the log records not yet written to the log by the participant available, thus ensuring resilience to failures. This approach uses multicast and has the same latency as 2PC, but requires more messages to be sent.

A third approach [28] is the most similar to the approach adopted in this paper. The differences are that it incurs unnecessary overhead by sending the “start of prepare” and the commit log records to the backup, and it forces log records to the disk even if both the primary and the backup work correctly. The performance is thus degraded.

3 System Model

The system is composed of a number of processes or nodes connected through a communication network. Each process has both a functional unit (application or database server) and a transaction manager. A process executes two kinds of actions. (1) Change state and (2) send or receive a message. When correct, they execute at arbitrary speeds, but eventually make progress. Processes fail by crashing, causing them to lose state. Such events are, however, rare. A failed process is recovered and brought up-to-date by the system.

Communication is *asynchronous* and *reliable*. Thus, there are no bounds on communication delays and messages are not corrupted or lost if both the receiving and the sending process behaves correctly, i.e. do not crash.

In an asynchronous system, a failure detector is needed to make the system reliable [29]. An *eventually strong* failure detector can solve the atomic commitment problem [30]. However, to simplify the problem descriptions and explanations a perfect failure detector that eventually suspects every faulty process and never suspects a correct process is assumed.

For the purpose of this paper no disks are used. State is stored entirely in main-memory. Thus, to make the state persistent and the system highly available the *primary-backup approach* [2] is used. This approach assumes that MTTF is orders of magnitude larger than MTTR, thus both the primary and backup do not fail at the same time.

Following [16], the costs of execution in this system are twofold. (1) The computation cost is the total number of messages sent, and (2) the delay is serialized messages. The main memory operations associated with the atomic commitment protocol are only a small fraction of the load on the system, thus their costs are assumed to be negligible. Also, as long as the processors are not fully utilized, there are no queueing effects.

4 The Non-Blocking Atomic Commitment Problem

An atomic commitment protocol ensures that the participants in a transaction agrees on the outcome, i.e. ABORT or COMMIT. Each participant vote, YES or NO, on whether they can guarantee the local ACID properties of the transaction. All participants has a right to *veto* the transaction, thus causing it to abort. The *Non-Blocking Atomic Commitment* problem, NB-AC, has these properties [1030]:

- NB-AC1.** *<uniform agreement>* All processes that decide reach the same decision.
- NB-AC2.** *<integrity>* A process cannot reverse its decision after it has reached one.
- NB-AC3.** *<uniform validity>* COMMIT can only be reached if *all* processes voted YES.
- NB-AC4.** *<non-triviality>* If there are no failures and no processes voted NO, then the decision will be to COMMIT.
- NB-AC5.** *<termination>* Every correct process eventually decides.

5 The Circular Two-Phase Commit Protocol

This section presents the *Circular Two-Phase Commit protocol*, C2PC, for main memory primary-backup systems.

Normally, 2PC requires both forced and non-forced disk writes [169]. In a primary-backup environment these disk writes can be replaced by, respectively, synchronous (blocking) and asynchronous (non-blocking) logging to the backup node. Figure 1(a) illustrates this. The small arrows between each primary-backup pair is the logging.

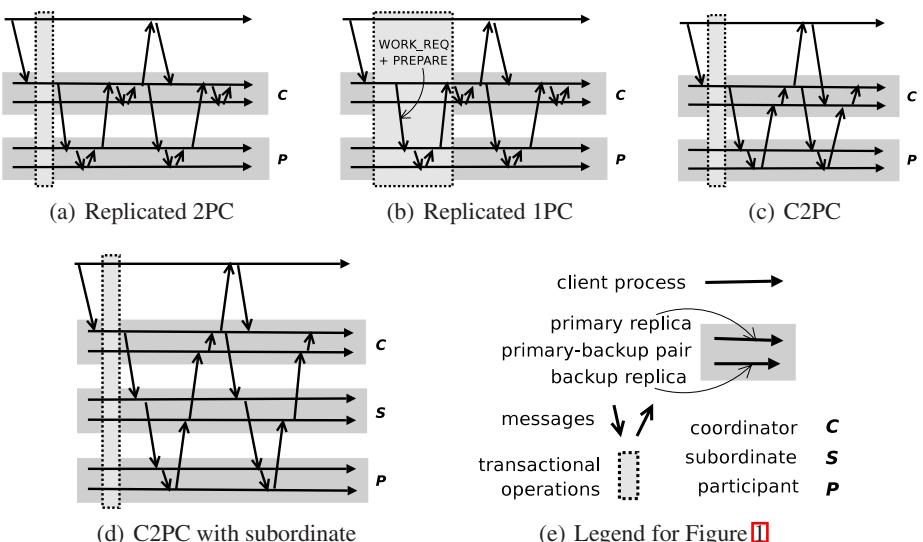


Fig. 1. Execution of various atomic commitment protocols

2PC (Figure 1(a)) consists of two phases, a voting phase and a decision phase. In the voting phase the votes are collected by a coordinator, and the coordinator makes a decision depending on the votes and persistently stores the decision. In the decision phase, the outcome is sent to the participants which send an acknowledgement back to the coordinator. Each participant must persistently store its vote and the outcome before replying to the coordinator in, respectively, the voting and decision phase. After the decision has been made persistent, the coordinator can give an *early answer* [3] to the client. Thus, the response time seen from the client is less than what it would be if the second phase had to be completed before the reply.

1PC (Figure 1(b)) piggybacks the prepare-message on the last work request for the transaction. Thus, the first phase of the voting is eliminated. However, each participant's vote must be persistently stored to the backups before replying to the coordinator.

The C2PC protocol is a modified version of 2PC for main memory primary-backup systems. Similarly to 2PC, C2PC has two phases and logs the votes and decision to the backups. However, it allows the backup to reply to the backup coordinator. This is shown in Figure 1(c). Instead of sending votes and acknowledgments back and forth the votes and decision are sent in a ring for each branch of the commit tree. This is a case of the transfer-of-commit optimization [9] where the authority to commit is passed via the participants to the backup root coordinator.

C2PC reduces both the number of messages in the critical path and the total number of messages to commit a transaction. The critical path is the delay until the transaction coordinator can give an early answer to the client. For instance, comparing Figure 1(a) and 1(c), the added delay has been reduced from six to four messages and the added number of messages from thirteen to nine. By comparison, 1PC (Figure 1(b)) has an added delay of four, two within the transactional operations frame and two after, and a total overhead of eleven messages.

During normal processing, the communication goes through each ring twice, one for each phase as seen in Figure 1(c). In the first round, the primary coordinator, *pc*, votes and piggybacks its own vote on the prepare message to the primary participant, *pp*. Each *pp* vote and sends its vote along with the vote of the *pc* to the backup participant, *bp*. *Bp* adds its own vote and forwards it to the backup coordinator, *bc*. *Bc* makes a decision based on the received votes and its own. The decision is then made persistent by sending it to the *pc*, which gives an early answer to the client and initiate the second phase.

The protocol also handles subcoordinators, or *subordinate* processes [6]. A subordinate acts as a participant to the coordinator and as a coordinator to the participants. A subordinate can also act as a participant to another subordinate. During the first phase a primary subordinate, *ps*, votes and forwards the vote to each of the subparticipants. The backup subordinate, *bs*, collects the votes from all the subparticipants before it sends its vote to the *bc*. During the second phase the decision is propagated in the same fashion.

If, during the first phase, one of the participants or subordinates votes No, the vote is propagated back to the *bc*, while each subordinate along the way makes the decision to abort. The decision is then sent out to all remaining undecided participants and subordinates.

The protocol handles failures of both the primary and the backup. These failure scenarios might occur:

- If one of the primaries fails during the first phase, the transaction is aborted as the backup cannot be sure that it has all the log records.
- If one of the backups fails during the first phase, the preceding node in the ring sends the vote message to the primary instead.
- If one of the participating primaries (resp. backups) fails during the second phase, the preceding node in the ring sends the decision or acknowledgement message to the backup (resp. primary) instead.

Rerouting the messages to the non-failed primary or backup in the last two scenarios above works since the primary and backup is assumed never to fail at the same time.

First, a detailed explanation is given, second, the correctness of the protocol is proven and, third, a one-phase version of C2PC, C1PC, is outlined.

5.1 Detailed Description

This section presents the C2PC protocol in detail. Listings 1.1 to 1.6 present the protocol in failure free scenarios for all types of nodes.

Each process has a Transaction Table (TT) which holds the state (*active*, *prepared*, *committed* or *aborted*) and known participants of each transaction. Also, it is told which processes have failed from the local failure detector. Log records marked with a Log Sequence Number (LSN) [IO] are shipped asynchronously to the backup. The backup checks that it has received all LSNs and acknowledges the greatest LSN received so far. The TT of the primary holds the greatest LSN acknowledged so far by the backup, and the backup TT is updated as log records are received and acknowledged from the primary. When voting, any unacknowledged log records are piggybacked on a `VoteMsg`. The TT can also be changed by receiving a vote message, `VoteMsg`, from a participant.

First, the protocols for the coordinator and the participants are presented. Then, the protocols for the subordinates are given.

Coordinator and Participants. As seen in Listing 1.1 the *pc* of the transaction initiates the protocol by attaching its own vote to a `VoteMsg` and sending it to each of the participants.

Some necessary information is included in all messages going down in the commit tree: (1) The transaction identifier, (2) the address of the primary and backup of the *pc* and (3) the address of the client. The first identifies the transaction to be committed, while the second allows *bp* to contact *bc*. The third allows *bc* to contact the client to complete the transaction should *pc* fail. Also, included in at least one of the vote messages are (4) the unacknowledged log records of the transaction at *pc* and (5) a list of the participants of the transaction. The fourth ensures that *bc* has all the log records generated by *pc* of the transaction before committing it. Finally, the fifth guarantees that *bc* waits for `VoteMsgs` from all the participants before making a decision and enables it to complete a transaction in case *pc* fails.

Each *pp* (Listing 1.3) of the transaction receives a `VoteMsg`. If the received vote or its own is NO, the decision is ABORT, and a new `VoteMsg` with a No-vote is sent to the

```

atomic.commitment: 1
if (myVote == NO) {
    decide(ABORT);
    voteMsg = new VoteMsg(txn,No);
    send (voteMsg) to all participants; 4
} else {
    voteMsg = new VoteMsg(txn,Yes); 7
    send (voteMsg) to all participants;
    receive(DecisionMsg) from backup {
        if (decision is COMMIT) decide(COMMIT); 10
        else decide(ABORT);
        dMsg = new DecisionMsg(txn,decision);
        send reply to client;
        send (dMsg) to all participants; 13
        if (decision is COMMIT) {
            receive (AckMsg) from backup; 16
            on timeout {resend dMsg;}
        }
    }
}
}

```

Listing 1.1. Primary coordinator

```

atomic.commitment: 20
receive (voteMsg) from all participants;
if (receivedVotes == NO || myVote == NO) {
    decide(ABORT);
    dMsg = new DecisionMsg(txn,ABORT); 23
} else {
    decide(COMMIT);
    dMsg = new DecisionMsg(txn,COMMIT); 26
}
send (dMsg) to primary;
if (decision is COMMIT) { 29
    receive (DecisionMsg) from all;
    receive ack from client;
    send (AckMsg) to primary;
}
}

```

Listing 1.2. Backup coordinator

```

atomic.commitment:
receive(voteMsg);
if (receivedVote == NO || myVote == NO) { 35
    decide(ABORT);
    vMsg = new VoteMsg(txn,NO);
    send (vMsg) to backup; 38
} else {
    txnLog = getLog(txn);
    vMsg = new VoteMsg(txn,vote,info); 41
    send (vMsg) to backup;
    receive(decisionMsg) {
        decide(decisionMsg.decision); 44
        send (decisionMsg) to backup;
    }
}

```

Listing 1.3. Primary participant

```

atomic.commitment: 47
receive (voteMsg);
if (receivedVote == NO || myVote == NO) {
    decide(ABORT);
    vMsg = new VoteMsg(txn,NO);
    send (vMsg) to parentBackup; 50
} else {
    vMsg = new VoteMsg(txn,YES); 53
    send (vMsg) to parentBackup;
    receive (decisionMsg) {
        decide(decisionMsg.decision);
        send (AckMsg) to parentBackup; 56
    }
}

```

Listing 1.4. Backup participant

backup. If the vote is YES, *pp* adds its unacknowledged log records for the transaction to the vote message and forwards it to the backup.

When a YES-vote is received by a *bp* (Listing 1.4), the log records from the *pp* are removed from the *VoteMsg* and applied to the local log. The local vote is then collected and forwarded to the backup of the parent. If the local vote or the received one is NO, the decision is ABORT and a *VoteMsg* containing a No-vote is forwarded to *bc*.

Listing 1.2 shows the algorithm for *bc*. Upon receiving a *VoteMsg*, it checks if the message contains a list of participants. If so, it checks whether or not it has received a *VoteMsg* from all of them. If a list is not included, it knows that there are more messages coming. Either way, it waits until all participants' votes have been collected (line 20), and then makes a decision: ABORT if any NO-votes have arrived or itself votes NO, otherwise COMMIT. Any unacknowledged log records sent from *pc* are appended to the local log and a decision message, *DMsg*, is then sent to *pc*.

When the *pc* receives a *DMsg*, it decides the same and then forwards the decision to the client and the participants. If the decision is COMMIT it waits to receive a confirmation from *bc* saying that all participants have committed before the transaction can be removed from TT.

```

atomic_commitment: 59
receive(voteMsg);
if (receivedVote == NO || myVote == NO){
    decide(ABORT); 62
    vMsg = new VoteMsg(txn,NO);
    send (vMsg) to participants;
} else { 65
    txnLog = getLog(txn);
    voteMsg = new VoteMsg(txn,vote);
    send (voteMsg) to participants; 68
    receive(decisionMsg) {
        decide(decisionMsg.decision);
        send (decisionMsg) to participants; 71
    }
}

```

Listing 1.5. Primary subordinate

```

atomic_commitment: 74
receive (voteMsg) from all subparticipants;
if (receivedVotes == NO || myVote == NO) { 77
    decide(ABORT);
    vMsg = new VoteMsg(txn,NO);
    send (vMsg) to parentBackup;
} else {
    vMsg = new VoteMsg(txn, YES); 80
    send (vMsg) to parentBackup;
    receive (decisionMsg) {
        decide(decisionMsg.decision);
        send (DecisionMsg) to parentBackup;
    }
}

```

Listing 1.6. Backup subordinate

After voting, the participants waits for a decision. When received, the decision is made, and the message is sent from the *pp* to the *bp* to the *bc*. Note that since the *pc* and the *bc* are assumed not to fail at the same time, a termination protocol is not needed for the participants, because the coordinator ensures the liveness of the transaction.

Subordinate Processes. The previous subsection is necessary to make an atomic commitment, but internal nodes in the commit tree can also exist. These nodes are called *subordinates* [16] and are characterized by acting as a coordinator for some participants, while being a participant itself for the coordinator or other subordinate.

The protocol for a primary subordinate, *ps*, is given in Listing 1.5. When a *VoteMsg* is received, it decides ABORT if the received or its own vote is NO. Otherwise, the unacknowledged log records are appended to at least one of the outgoing *VoteMsgs* along with a list of the participants. Either way, the address of the current *ps* and *bs* is sent to the participants along with the vote and the information received in the *VoteMsg*.

A backup subordinate, *bs*, (Listing 1.6) waits, as the *bc*, until a *VoteMsg* is received from all its participants and then makes a decision based on the received vote and, if all votes are YES, the result of applying the log records received from the primary. The information from the parent primary is added to the *VoteMsg*, and it is sent to the parent backup.

In the same way as the participants, the subordinates waits for a decision after voting. When received, the decision is made, and the message is sent from the *ps* to a *pp* or another *ps*. The *bs* receives the decision from one or more *bps* or *bss*, and forwards it to the *bc*. For the same reasons as for the participants, a termination protocol is not needed here.

5.2 Correctness

This section proves the correctness of the C2PC protocol by proving each of the properties given in Section 4 in this order: **NB-AC2**, **NB-AC3**, **NB-AC4**, **NB-AC1** and **NB-AC5**.

Lemma 1. NB-AC2: *A process cannot reverse its decision after it has reached one.*

Proof. The algorithms for each of the processes use if-else statements to avoid deciding more than once per process.

Lemma 2. NB-AC3: *The COMMIT decision can only be reached if all processes voted YES.*

Proof. All processes can decide COMMIT during the second phase of the protocol. However, they can only decide COMMIT if they receive a message with a COMMIT decision. The only process that can decide COMMIT during the first phase is bc (line 25). This happens only if it has received YES-votes from all the participating processes including itself.

Lemma 3. NB-AC4: *If no process failed and no process voted No, then the decision will be to COMMIT.*

Proof. If no process failed, and no process voted NO, then since the communication system is reliable, bc receives YES from all participants and subordinates. Thus, COMMIT is reached (line 25).

Lemma 4. NB-AC1: *All processes that decide reach the same decision.*

Proof. A process can only decide ABORT during the second phase, if a process decided ABORT during the first phase. Similarly, a process can only decide COMMIT during the second phase, if bc decided COMMIT during the first phase. As proved in Lemma 2, COMMIT can be decided (line 25) only if all processes voted YES. A process can only decide ABORT during the first phase if it votes NO. A process cannot both vote YES and NO, so two processes cannot decide differently.

Lemma 5. NB-AC5: *Every correct process eventually decides.*

Proof. To enable a process to decide in the presence of failures, all failure scenarios as well as the scenario with no failures must be handled. These scenarios can occur:

- 1: Pc fails before sending the vote to all participants.
- 2: Pc fails after initiating the voting, but before sending the decision to all participants.
- 3: Pc fails after sending the decision to all participants, but before receiving an AckMsg from bc .
- 4: Bc fails before sending the decision to pc .
- 5: Bc fails after sending the decision to pc , but before sending AckMsg to pc .
- 6: A ps , bs , pp or bp fails before sending the vote.
- 7: A ps , bs , pp or bp fails after sending the vote, but before sending the decision.
- 8: No node fails.

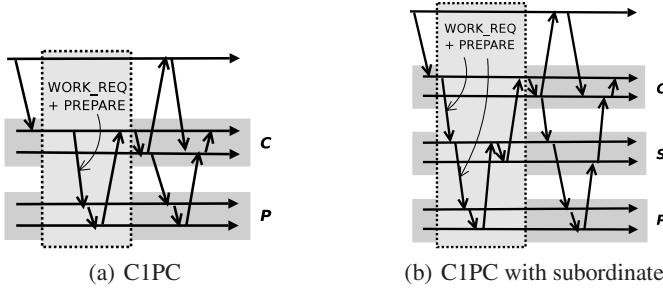
Scenario (1): None has voted, each of the participants can independently abort the transaction after a timeout has expired without causing inconsistencies in the system.

Scenario (2): When bc does not receive a decision from any of the participants and pc fails, bc can complete the transaction with the decided outcome.

Scenario (3): The AckMsg is sent to pc to allow it to purge the transaction entry from its TT. However, this is not needed if pc fails, because it will have to update its TT as part of the recovery process.

Scenarios (4) and (6): If pc does not receive a decision within a given time limit it can send a message to bc and tell it about the timeout, then bc can decide Abort. If bc has failed, pc can safely abort the transaction.

Scenario (5): The transaction is completed, but if the decision is COMMIT the transaction entry will not be deleted from the TT of pc until an AckMsg is received. However,

**Fig. 2.** Examples of C1PC execution

pc resends the decision (line 17) with updated backup information until it receives confirmation that all participants have decided.

Scenario (7): When a process fails during the second phase, the decision must be sent via the backup on its way down the commit tree or via the primary on its way up the tree. Pc resends the decision (line 17) until it receives an acknowledgement, and the failed processes are bypassed.

Scenario (8): This is proven similarly to Lemma 3. Since no process failed and the communication system is reliable, bc receives votes from all participants and subordinates. Thus, it decides either COMMIT in line 25 or ABORT in line 22. By the same argument each participant and subordinate eventually decides.

All scenarios are handled, thus, all correct processes eventually decides.

Theorem 1. *C2PC is a valid non-blocking atomic commitment protocol.*

Proof. Since C2PC satisfies properties **NB-AC1 - NB-AC5** it solves NB-AC. ■

5.3 C1PC

Circular One-Phase Commit protocol, C1PC, is a circular version of 1PC and can be designed as shown in Figure 2. The main differences between C1PC and C2PC are: During the first phase (1) pc piggybacks `VoteMsg` on the last request and (2) bp replies to ps or pc (instead of bs or bc) because there might be results that are needed. During the second phase, (3) pc makes the decision to commit, and (4) bc replies to the client and sends the `DMsg` to the participants.

6 Evaluation

This section compares the performance of non-fault tolerant, replicated 2PC, replicated 1PC, C2PC and C1PC. We assume the normal operational mode where no participating processes fails and all participants vote YES. The purpose is to evaluate the costs associated with the various protocols.

The table in Figure 3 shows formulas for the added number of messages in the critical path and the total overhead to complete a transaction compared to the non-fault tolerant case. The critical path is the delay until the transaction coordinator can give an early answer to the client. Parallel and linear execution corresponds to a commit-tree of height 1 and $N - 1$ respectively.

Protocol	Delay	Total
Non-fault tolerant	0	0
Replicated 2PC, parallel	6	$8N + 5$
Replicated 2PC, linear	$4N + 4$	$8N + 5$
Replicated 1PC, parallel	4	$6N + 5$
Replicated 1PC, linear	$2N + 2$	$6N + 5$
C2PC, parallel	4	$8N + 1$
C2PC, linear	$2N + 1$	$4N + 5$
C2PC, hybrid	4	$6N + 1$
C1PC, parallel	2	$4N + 3$
C1PC, linear	$N + 1$	$3N + 4$
C1PC, hybrid	2	$3N + 4$

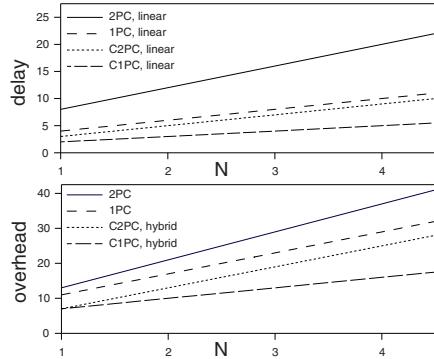


Fig. 3. Added delay until early answer to client and total overhead for various ACPs. $N = \#$ servers invoked by transaction excluding the coordinator, $N \geq 1$

The non-fault tolerant case is non-replicated and has zero delay and overhead to complete the request. It does not tolerate any failures and there is no coordination of the outcome.

For the transactional cases, the parallel versions of C2PC and C1PC have the shortest delay and the linear versions have the least overhead. This observation leads to the *hybrid* versions of C2PC and C1PC, where the voting phase is executed in parallel and the decision phase in linear. This minimizes both the delay and the overhead.

The graphs in Figure 3 depicts the delay and overhead of selected protocols. The protocols with constant delay are not shown in the delay graph and in the overhead graph the linear and parallel circular protocols are not showed to avoid cluttering.

The delay of parallel and hybrid C2PC is equal to and two-thirds of the delay of 1PC and 2PC, respectively. C1PC halves the delay and almost halves the overhead compared to 1PC, but also inherits its restrictions and assumptions [22]. The overhead of the parallel and hybrid versions of C1PC is almost half of that of 1PC, and hybrid C2PC has less overhead than 1PC.

7 Conclusion

This paper has presented an atomic commitment protocol, Circular Two-Phase Commit (C2PC). It is a single fault-tolerant optimization of 2PC for replicated main-memory primary-backup systems. C2PC does not require any changes to the standard 2PC interface, and can be implemented in an asynchronous system with an unreliable failure detector. The protocol is unique in the sense that it does not log to disk and ensures liveness for both data, processing and transaction commitment.

For further work the protocol should be implemented and performance measures should be made to verify the analysis and evaluation in Section 6.

References

1. Schneider, F.B.: Replication management using the state machine approach. In: Distributed systems (2nd Ed.). ACM Press/Addison-Wesley Publishing Co. (1993) 169–197
2. Budhiraja, N., Marzullo, K., Schneider, F.B., Toueg, S.: Distributed systems. In Mullender, S., ed.: Distributed Systems. ACM Press. second edn. Addison-Wesley (1993) 199–216
3. et al., S.O.H.: The ClustRa telecom database: High availability, high throughput, and real-time response. In: Proc. of VLDB. (1995)
4. Härdér, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Comput. Surv. **15** (1983) 287–317
5. Gray, J.: Notes on data base operating systems. In: Operating Systems, An Advanced Course, London, UK, Springer-Verlag (1978) 393–481
6. Spiro, P.M., Joshi, A.M., Rengarajan, T.K.: Designing an optimized transaction commit protocol. j-DEC-TECH-J **3** (1991) 70–78
7. Lampson, B., Lomet, D.: A new presumed commit optimization for two phase commit. In: Proc. of VLDB. (1993)
8. Haritsa, J.R., Ramamritham, K., Gupta, R.: The prompt real-time commit protocol. IEEE Trans. Parallel Distrib. Syst. **11** (2000) 160–181
9. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann (1993)
10. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency control and recovery in database systems. Addison-Wesley Longman Publ. Co., Inc. (1986)
11. Reddy, P.K., Kitsuregawa, M.: Reducing the blocking in two-phase commit protocol employing backup sites. In: Proc. of CoopIS. (1998)
12. Skeen, D.: Nonblocking commit protocols. In: Proc. of SIGMOD. (1981)
13. Rabinovich, M., Lazowska, E.D.: A fault-tolerant commit protocol for replicated databases. In: Proc. of PODS. (1992)
14. Guerraoui, R., Larrea, M., Schiper, A.: Reducing the cost for non-blocking in atomic commitment. In: (ICDCS), Hong Kong (1996) 692–697
15. Samaras, G., Britton, K., Citron, A., Mohan, C.: Two-phase commit optimizations and trade-offs in the commercial environment. In: Proc. of ICDE. (1993)
16. Mohan, C., Lindsay, B., Obermarck, R.: Transaction management in the R* distributed database management system. ACM Trans. Database Syst. **11** (1986) 378–396
17. Stamos, J.W., Cristian, F.: A low-cost atomic commit protocol. In: Proc. of SRDS. (1990)
18. Gawlick, D., Kinkade, D.: Varieties of concurrency control in IMS/VS Fast Path. IEEE Database Eng. Bull. **8** (1985) 3–10
19. Park, T., Yeom, H.Y.: A consistent group commit protocol for distributed database systems. Proc. of PDCS (1999)
20. Levy, E., Korth, H.F., Silberschatz, A.: An optimistic commit protocol for distributed transaction management. In: Proc. of SIGMOD. (1991)
21. Abdallah, M., Pucheral, P.: A single-phase non-blocking atomic commitment protocol. In: Proc. of DEXA. (1998)
22. Abdallah, M., Guerraoui, R., Pucheral, P.: One-phase commit: Does it make sense? In: Proc. of ICPADS, Washington, DC, USA (1998)
23. Lee, I., Yeom, H.Y.: A single phase distributed commit protocol for main memory database systems (2002)
24. Stamos, J.W., Cristian, F.: Coordinator log transaction execution protocol. Distributed and Parallel Databases **1** (1993) 383–408
25. Stonebraker, M.: Concurrency control and consistency of multiple copies of data in distributed ingres. IEEE Trans. Software Eng. **5** (1979) 188–194
26. Reddy, P.K., Kitsuregawa, M.: Blocking reduction in two-phase commit protocol with multiple backup sites. In: DNIS. (2000)

27. Jiménez-Peris, R., Patiño-Martínez, M., Alonso, G., Arévalo, S.: A low-latency non-blocking commit service. In: Proc. of DISC. (2001)
28. Mehrotra, S., Hu, K., Kaplan, S.: Dealing with partial failures in multiple processor primary-backup systems. In: Proc. of CIKM. (1997)
29. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43** (1996) 225–267
30. Guerraoui, R.: Revisiting the relationship between non-blocking atomic commitment and consensus. In: WDAG. (1995)

Towards Timely ACID Transactions in DBMS

Marco Vieira¹, António C. Costa², and Henrique Madeira¹

¹ CISUC - University of Coimbra, Polo II,
3030 Coimbra, Portugal

{mvieira, henrique}@dei.uc.pt
² FC – University of Lisbon,
Lisbon, Portugal

casim@di.fc.ul.pt

Abstract. On-time data management is becoming a key difficulty faced by the information infrastructure of most organizations. In fact, database applications for critical areas are increasingly giving more importance to the timely execution of transactions. Database applications with timeliness requirements have to deal with the possible occurrence of timing failures, when the operations specified in the transaction do not complete within the expected deadlines. In spite of the importance of timeliness requirements in database applications, typical commercial DBMS do not assure any temporal properties, not even the detection of the cases when the transaction takes longer than the expected/desired time. This paper discusses the problem of timing failure detection in database applications and proposes a transaction programming approach to help developers in programming database applications with time constraints. The paper illustrates the proposed programming model with a practical example using the Oracle 10g DBMS running a performance benchmark for real-time database applications.

Keywords: Databases, transaction processing, performance, timely transactions.

1 Introduction

Developing database applications with timeliness requirements is a very difficult problem as current database technology does not provide easy programming support that help engineers and programmers in dealing with timing issues. This is true for all the programming layers of a typical database application: the database management system (DBMS), the middle layer software (e.g., web-server, application-server, etc), and the client application. Nevertheless, real database applications very often have to cope with the possible occurrence of timing failures, when the operations specified in a transaction do not complete within the expected deadlines. Without adequate support to help designers and programmers to solve timing requirements, the development of these applications is a very complex task.

The notion of time is completely absent from the classical DBMS transactional model, which is based on the ACID properties (Atomicity, Consistency, Isolation, and

Durability) [1]. [2] proposes that research should be conducted in order to add timeliness properties to the typical ACID properties, which would provide to the application layer timely ACID properties or, in short, TACID. In this paper, which represents the first step towards TACID transactions, we discuss the problem of timing failure detection in database applications and propose a transaction programming approach to help developers in programming database applications with time constraints. According to the timing requirements we classify applications in different classes, namely: traditional (no temporal requirements), fail-safe, time-elastic, and fail-operational. To implement these classes we propose the following types of transactions, which support different temporal requirements: transactions with no temporal requirements (typical ACID transactions), transactions with strict temporal requirements, and transactions with relaxed temporal requirements.

This paper proposes a new approach for transaction programming, which allows concurrent detection of timing failures during execution, including for distributed transactions. Timing failure detection can be performed at the database clients' interface, in the database server, or in a distributed manner. An application programming interface (API) that implements this new transaction programming approach is provided. It can be used by database programmers to easily implement applications with timeliness requirements. All source code, including an example of utilization, is available at [<http://gbd.dei.uc.pt/downloads.php>] for public use.

The structure of the paper is as follows. The following section discusses the problem of timing failure detection and proposes a classification for database transactions and applications. Section 3 proposes a new transactions programming approach and presents the application programming interface developed. Section 4 presents the experimental evaluation and Section 5 concludes the paper.

2 Timeliness Requirements in Database Applications

In many situations timeliness is as important as correctness and atomicity. For instance, in a database application designed to manage information about a critical activity, e.g., air traffic control, a transaction that reads and stores the current reading from a positioning radar must be executed in a short time (i.e., the longer it takes to execute the transaction the less useful the reading is). In other words, in many applications, a transaction that does not complete before a specified deadline becomes useless and this situation must be reported to the application/business layer in order to be handled in an adequate manner. But worse than becoming useless, the delayed execution of a transaction can compromise safety properties of a system. In such cases, the detection of a violated deadline would allow the execution of fall-back or recovery actions, isolating and avoiding the propagation of the failure to other components and, consequently, the occurrence of more severe failures.

Real-time databases have emerged some years ago. However, these databases have been designed and implemented for very specific applications [3], [4]. To support real-time applications, real-time databases relax the ACID requirements to allow better support for temporal consistency while maintaining support for data consistency [5], [6]. Semantic information is used to determine to what degree the ACID properties must be enforced.

In real-time DBMS the ACID properties are normally applied only to parts of the transaction. Nevertheless, important features such as timing failure detection or, more generically, timing fault-tolerance, have been completely neglected, which also restricts the application areas for such DBMS. The problem is even worse if we consider the possibility of deploying distributed DBMS over wide-area or open environments. Such environments exhibit poor baseline synchrony and reliability properties, thus making it more difficult to deal with timeliness requirements. Obviously, this uncertainty and lack of timeliness will directly affect the execution of transactions, which, as an immediate effect, will be delayed. However, more severe effects may also be observed on the account of timing failures.

The environments we consider in this paper can be characterized, essentially, as environments of partial synchrony. In fact, their basic synchrony properties are only cluttered from time to time, or by specific parts of the structure. Several partial synchrony models have been proposed, with different solutions to address application timeliness requirements. The idea of using failure detectors that encapsulate the synchrony properties of the system was first proposed in [7]. The work in [8] introduces the notion of Global Stabilization Time (GST), which limits the uncertainty of the environment. The Timed model, proposed in [9], allows the construction of fail-aware services, which always behave timely or else provide awareness of their failure.

Our proposal is to bring timing failure detection to the typical ACID transactions implemented by most commercial DBMS, putting together classic database transactions management and distributed timely computing. The goal is to extend the typical transaction programming approach in order to support transactions with ACID properties together with timing failure detection.

In order to add timing failure detection to the typical ACID transactions, we propose that the basic toolset to be offered to database application programmers should include the following classes of transactions:

- **Transactions with no temporal requirements:** Typical ACID transactions implemented by classic database management systems. The database clients do not expect any timeliness guarantees, not even the detection of timing failures.
- **Transactions with strict temporal requirements:** For this class, the database clients can specify a time frame in which the transaction has to be concluded to succeed. In this class, the system must at least provide timing failure detection, even in distributed transactional environments. The transaction is rolled back if the last command in the transaction is not executed before the specified deadline and the client application is notified in order to cope with the occurrence of the timing failure.
- **Transactions with relaxed temporal requirements:** In this class, the transactions are always executed independently of the specified time frame. However, if the deadline is reached before the transaction commits, the client application is nevertheless informed. This allows the application to execute any task related to the occurrence of the timing failure (e.g., notify the DBA) and continue the execution of the transaction.

Real database applications very often have to deal with different timing requirements, whose execution must be supported by one or more of the classes of

transactions proposed before. The following points present our classification for database applications concerning timing constraints, and give some examples of applications from real scenarios:

- **Traditional applications class:** Typical applications with no temporal requirements.
- **Fail-safe class:** Applications that can switch to a fail-safe state when there is a timing failure. When a transaction is submitted the application waits for the transaction response or a notification that a timing failure has occurred. The application must be informed about a timing failure occurrence as soon as the deadline specified for the transaction is exceeded. In this case the application executes some conforming actions and enters a fail-safe state. These database applications can be implemented based on transactions with strict temporal requirements. Typical examples include manufacturing industrial processes (electronic industry, automotive manufacturing industry, etc) where it is possible to stop the manufacturing chain in case of delay in the database transaction execution (that fail-safe state is in general necessary because of mechanical issues of manufacturing processes).
- **Time-elastic class:** Applications able to adapt timing constraints during execution. In this case, the collection of information about timing failures and the temporal execution of transactions can be used to feed a monitoring component or to tune specific application parameters in order to adapt its behavior to the actual load conditions of the system. The application may decrease the transaction submission rate, increase the transactions deadline if possible, or postpone the execution of transactions to a latter time. In environments with replicated databases, the application can also perform load balancing based on timing failure detection. These applications can be implemented based on transactions with strict temporal requirements or based on transactions with relaxed temporal requirements. Examples of this class include databases that control mobile communication systems, where connection establishment can tolerate some delays (or may be refused) and billing transactions can be postponed, or continuous manufacturing processes such as chemical processes.
- **Fail-operational class:** Applications that continue executing transactions without adapting timing constraints during execution, regardless of the timing failures detected. The client application should be notified about the occurrence of the timing failure but the execution of the transaction does not stop. Timing failure detection is used by the application to perform specific actions (e.g., notify the database administrator) when the execution of transactions exceed the deadline. These applications can be implemented based on transactions with strict temporal requirements or transactions with relaxed temporal requirements. Examples of this class include pay-per-view television applications and video streaming.

3 New Transactions Programming Approach

Transactions in typical database application are executed by submitting one command at a time or using batches of commands. After submitting a command (or a batch of commands) the client application waits for the corresponding response. The database server can be a single machine or a distributed set of replicated servers. An important aspect is that neither the client layer nor the database server are concerned about the detection of timing failures.

In order to provide transactions with temporal requirements we need a new approach for transaction programming. Our proposal is to modify the typical database environment in order to include timing failure detection capabilities. Three possibilities can be considered: detection in the client layer, detection in the database server, and distributed detection. To measure the duration of both local and distributed actions we have adopted the Timely Computing Base (TCB) model [10], which is based on an improved round-trip technique that guarantees not only bounded, but also almost stable measurement errors. Informally speaking, the TCB Distributed Duration Measurement service can be used for monitoring the duration of local or distributed actions. It is the responsibility of the application to indicate the beginning and the end of the action. Local actions are measured in the same site, and in the case of distributed actions the measurement is made in a distributed way, using different TCB instances.

3.1 Timing Failure Detection in the Client Database Interface Layer

In a typical database environment the client application communicates with the server through an interface layer (e.g., Oracle Call Interface). This layer is specific for each DBMS and is responsible for managing all the communication with the database server. The detection of timing failures in the client requires the modification of this layer. As changing the database interface layer itself is difficult (this is normally a piece of proprietary software) our proposal is to add a wrapping layer through which all the communications between the client application and the database interface layer must go by. We call this layer Transactions Timing Failure Detection (TTFD) layer.

Figure 1 shows the architecture we propose for timing failure detection in the client interface layer. In order to implement timing failure detection capabilities, we propose the use of two connections between the client application and the TTFD layer: one to submit commands and receive results and the other to control timing definitions (e.g., the deadline and the type of the transaction) and receive timing failure notifications. These notifications are sent in the form of exceptions that must be handled by the client application. Note that, the TTFD layer can be used to abstract any particular implementation of a timing failure detection service. Therefore, it is possible to generate timing failure notifications at this layer, independently of the specific notification mechanism provided by the specific TTFD service implementation.

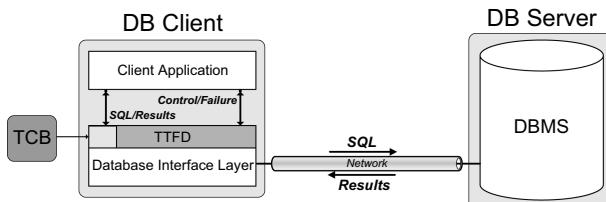


Fig. 1. Environment for timing failure detection in the database interface layer

When the client application begins a time critical transaction (i.e., submits the first SQL command in the transaction) and provides the class of the transaction (strict temporal requirements or relaxed temporal requirements) and the deadline for the

execution, the TTFD layer starts counting the elapsed time. The Duration Measurement service of the TCB is used for time measurement. The time critical transaction ends when the application executes a commit or rollback. If the deadline is violated before the end of the execution of the transaction an exception is generated and thrown to the client. If the transaction has strict temporal requirements it is automatically rolled back, otherwise, the transaction execution continues normally.

From the client point-of-view, the measured execution times includes the delays due to client-server network communication. This means that some false positives may occur due to the extra time that it takes for the response to travel from the server to the client. Timing failure detection in the database interface layer should be used in scenarios where it is important to take in consideration the time the client application waits for the response to the last command in the transaction (e.g., when the client application is able to perform load balancing at the network level). However, other solution is needed in scenarios where timeliness requirements apply to timed executions that terminate on the server side.

3.2 Timing Failure Detection in the Database Server

The second approach we propose consists on detecting timing failures at database server side. In this case the transactions execution time is measured from the server point-of-view. To provide the detection of timing failures in the database server we need to modify the DBMS implementation or to use a DBMS proxy that intercepts all the messages arriving to and leaving from the database server (see Figure 2). The database proxy also implements timing failure detection and the TCB is used to measure time. In the client side we also need the layer through which all the communications between the client application and the database interface layer must go by (TTFD layer). In this architecture this layer does not deal with timing failure detection. It receives timing definitions from the client application and forwards them to the database proxy. When the proxy detects a timing failure it notifies the TTFD layer, which raises the corresponding exception to the client application.

An important aspect is that all the communication between the TTFD layer and the DBMS proxy is performed using a communication channel different from the one used to send the SQL commands and results. The timing definitions and timing failure notifications are sent using this channel. Figure 2 presents the environment that we propose for timing failure detection in the database server. As mentioned before, the database server can be a single machine or a set of replicated servers.

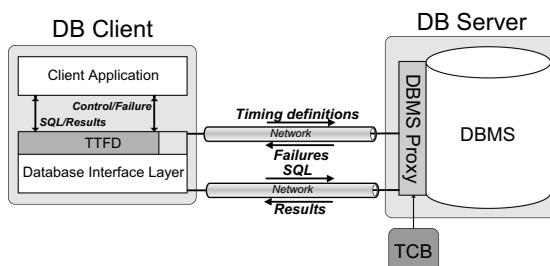


Fig. 2. Environment for timing failure detection in the database server

When the client application begins a time critical transaction (i.e., submits the first SQL command) and provides the class of the transaction and the deadline for the execution, the DBMS proxy starts counting the elapsed time using the TCB. If the deadline is exceeded before the end of the execution of the transaction an exception is sent to the client application through the TTFD layer. Transactions with strict temporal requirements are automatically rolled back. On the other hand, for transactions with relaxed temporal requirements the execution continues normally. Note that, rolling back the transaction is not influenced by any timing constraints as the state of the database only changes if the transaction commits.

Timing failure detection from the server point-of-view does not consider the amount of time that goes from the moment when the client submits the first command and the moment the server receives that command. This means that the transaction execution time is counted only after the first command is received by the DBMS proxy. Thus, if the communication between the client and the server is slow, there are some cases where a timing failure is not detected because the client/server communication time is not considered. Timing failure detection in the database server is useful in scenarios where the network delays should not be taken into account or are always so small that, from the client application point-of-view, have no impact in the transaction execution. It may also be useful if one decides to enrich the database server with real-time modules that are autonomously and immediately executed by the TCB upon failure detection.

3.3 Distributed Detection of Timing Failures

A transaction starts when the client application submits the first command and ends immediately after the server finishes the execution of the last command (and not when the client application receives the response). Thus, some database applications may require the execution time to be counted from the moment when the client submits the first command and the moment when the execution of last command ends at the server side. In this case the two solutions proposed before cannot be applied. It is necessary to use a form of timing failure detection based on distributed duration measurements. We will simply call it distributed timing failure detection.

An obvious problem raised by this approach is the distributed measurement of time. As it is well known, it is quite difficult to have synchronized clocks in different machines. To solve this problem we have decided to use the Duration Measurement service of the TCB model [10]. This service obviously requires local clocks of TCB modules to be read, and timestamps to be used and disseminated among relevant nodes of the system. Unlike the measurement of local actions, measuring distributed durations is quite more difficult because simply reading the clocks to get two timestamps is not sufficient. The distributed duration measurement service of the TCB is based on an improved round-trip technique [10] that guarantees not only bounded, but also almost stable measurement errors.

As shown in Figure 3, to provide distributed detection of timing failures we need to modify the DBMS implementation or to use a DBMS proxy and to include an additional layer in the database client that handles all the communications between the client application and the database interface layer (TTFD layer). This layer does not detect timing failures. It receives timing definitions from the client application and

instructs the TCB to start measuring the time. Timing definitions are sent to the DBMS proxy that detects timing failures using the distributed duration measurement capabilities of the TCB. When a timing failure is detected the DBMS proxy notifies the TTFD layer, which raises the corresponding exception to the client application.

As in the solutions proposed before, two connections are needed between the client application and the TTFD layer and the communication between the TTFD layer and the DBMS proxy also uses a communication channel different from the one used to send the SQL commands and receive the responses.

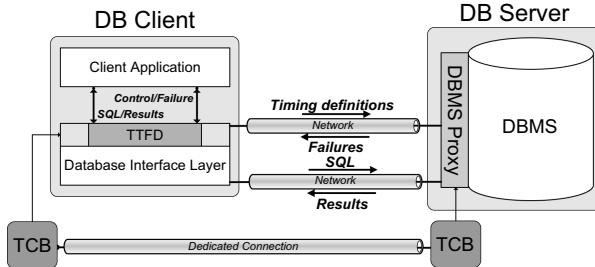


Fig. 3. Environment for distributed timing failure detection

3.4 Transactions Programming Interface

In order to allow database programmers to easily implement applications with timeliness requirements we have developed an application programming interface (API) that implements the new transaction programming approach proposed in this paper. In this work we have implemented this interface for JAVA and we are now starting to implement it for other languages (C++ and Delphi). An important aspect is that this API is as close as possible to the ones normally used by the programmers when implementing typical database applications. In fact, we tried to implement classes and methods similar to the ones typically used by programmers in terms of the names, parameters, and form of use (see example in section 4). Table 1 presents the most important methods provided.

4 Practical Example of Implementation

The example presented in this section has resulted from the study meant to demonstrate and evaluate the usefulness of the transactions programming approach proposed. As we are particularly interested in the aspects related to transaction execution time, we have decided to use a performance benchmark for real-time database systems for telecommunications [11]. This benchmark represents a telecommunications operator that includes several service providers, each one having its own database in a distributed environment.

The benchmark has been implemented using both the standard Java Database Connectivity (JDBC) API and the API proposed in this paper (considering both strict temporal requirements and relaxed temporal requirements). The first goal is to evaluate the effort needed to migrate from the traditional approach to the approach

Table 1. API provided for JAVA programmers. All source code, including an example of utilization, is available at [<http://gbd.dei.uc.pt/downloads.php>] for public use.

TACID Method	JDBC Method	Short Description
TACIDConnection getConnection(String URL, int approach)	Connection getConnection(String url)	Establishes a new TACID connection to the database. <i>url</i> represents the name of the database. <i>approach</i> is the timing failure detection level (client, server, or distributed)
void close() throws SQLException	void close() throws SQLException	Closes the database connection
void commit() throws SQLException, TACIDTimeoutException	void commit() throws SQLException	Commits the transaction. Throws an timeout exception if a timing failure has occurred meanwhile
void rollback() throws SQLException, TACIDTimeoutException	void rollback() throws SQLException	Rollbacks the transaction. Throws an timeout exception if a timing failure has occurred meanwhile
ResultSet startTimeQuery(int type, int timeout, String sql) throws SQLException, TACIDTimeoutException	ResultSet startTimeQuery(int type, int timeout, String sql) throws SQLException	Starts a new time critical transaction. SQL represents the first command (select command) in the transaction. <i>type</i> is the type of the transaction (strict or relaxed requirements). <i>timeout</i> represents the deadline for the transaction Returns the result of the command in a result set
int startTimeUpdate(int type, int timeout, String sql) throws SQLException, TACIDTimeoutException	int executeUpdate(String sql) throws SQLException	Starts a new time critical transaction. SQL represents the first command (insert, update or delete command) in the transaction
ResultSet executeQuery(String sql) throws SQLException, TACIDTimeoutException	ResultSet startTimeQuery(int type, int timeout, String sql) throws SQLException	Executes a query. The time critical transaction has already been started by another command
int executeUpdate(String sql) throws SQLException, TACIDTimeoutException	int executeUpdate(String sql) throws SQLException	Executes an insert, update, or delete command. The time critical transaction has already been started by another command
ResultSet getResultSet()	—	Returns the result of the last query executed
int getResultUpdate()	—	Get the result of the last insert, update, or delete executed
long getExecTime()	—	Returns the execution time for the last transaction

proposed in this paper. Table 2 presents a simple example of the use of timing failure detection in the benchmark.

As we can see, the TACID implementation is very similar (in both structure and commands) to the typical implementation, which facilitates the database programmers work. During the benchmark implementation we have observed a similar implementation time for both approaches. In fact, an experienced programmer takes around two days for each implementation. Obviously this implementation time depends strongly on the programmer's experience.

In order to evaluate the efficiency of the time failure detection approaches, we have performed several experiments. The basic experimental platform consists of three machines. Two machines are used as database servers and one as database client. The machines are connected using two dedicated fast-Ethernet networks. One is used for the SQL/results communication and the other is used by the TCB. Six service providers are considered, by implementing three databases in each database server.

The Oracle™ DBMS is one of the leading databases in the market and as one of the most complete and complex database it represents very well all the sophisticated relational DBMS available today. For that reason we have chosen Oracle 10g [12], which has been tuned based on the results from a previous work on the evaluation of the Oracle performance and recoverability [13].

The performance benchmark used has been implemented using the traditional approach (no timing failure detection) and considering timing failure at the three layers (clients' interface, database server, and distributed). Both transactions with strict temporal requirements and transactions with relaxed temporal requirements have been implemented. The performance benchmark was executed five times for each configuration (a total of 35 runs) with a duration of 10 minutes for each run.

Table 2. TACID implementation vs typical implementation: excerpt from the update subscriber transaction. The TACID implementation uses transactions with strict temporal requirements.

TACID Implementation	Typical Implementation
<pre> ... // Establish the connection Class.forName(driverName); con = TACIDdriverManager.getConnection(db, distributedDetection); try { // First command (timeout 100ms) sql="select addr from profile where sid='"+sid; rs=con.startTimeQuery(strictTempReq,100,sql); ... sql="update profile set addr='Coimbra' where sid='"+sid; recCount = con.executeUpdate(sql); con.commit(); } catch(TACIDTimeoutException e) { ... } ... </pre>	<pre> ... // Establish the connection Class.forName(driverName); con = DriverManager.getConnection (db); Statement con = con.createStatement(); try { // First command sql="select addr from profile where sid='"+sid; rs=con.executeQuery(sql); ... sql="update profile set addr='Coimbra' where sid='"+sid; recCount = con.executeUpdate(sql); con.commit(); } </pre>

Results have shown that timing failure detection does not introduce any overhead in transactions execution. For example, for the baseline configuration we have observed an average of 3769.9 transactions per minute (with a standard deviation of 23.17 transactions), while for timing failure detection at the client interface layer we have observed an average of 3784.5 transactions per minute (with a standard deviation of 38.96 transactions). The small deviations in the measures in successive runs are normal and just reflect the asynchronous nature of transactions. For the other layers (server and distributed) similar results have been observed.

Concerning execution time, the average using the baseline configuration was of 46.62 milliseconds (with a standard deviation of 0.25 milliseconds), while the average with timing failure detection at the client interface layer is around 41.58 milliseconds (with a standard deviation of 0.35 ms). This shows that the execution time when using time failure detection at the client interface layer is lower than the one observed for traditional transactions. This is due to the fact that when a transaction exceeds the deadline it is immediately rolled back and the client application continues the execution to the next transaction. Similar results were observed for the other layers.

Figure 4 presents an example of the execution profile for one of the transactions (*roaming user*) during one run of the benchmark using timing failure detection at the client interface layer and strict temporal requirements (similar profiles were observed for the other layers of timing failures detection). As we can see some transactions exceed the deadline, however in all the cases the timing failure was detected and the client application notified. These transactions are automatically rolled back. As show

in the figure, the transactions that exceed the deadline are detected a little bit after the deadline. This is due to the small latency of the detection mechanism (less than 20 milliseconds). Note that, database applications are characterized by long execution times (in some cases several seconds), thus a latency of 20 ms is quite acceptable.

Another important aspect is that, by the analyses of the results shown in Figure 4 we can see that even in sophisticated DBMS like Oracle 10g it is quite difficult to predict the execution time of the transactions. In fact, although most of the transactions are executed before the deadline some of them exceeded that deadline. This is due to many reasons such as the cache behavior, checkpointing, logging, etc. This demonstrates the importance of timing failure detection in database applications.

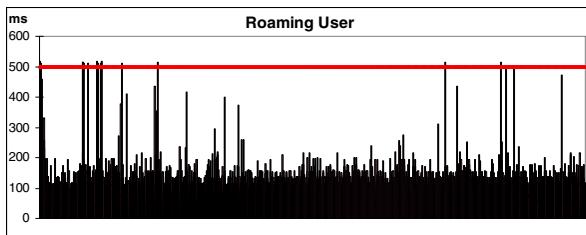


Fig. 4. Execution profile using timing failure detection. The horizontal line represents the deadline (500 ms) and each vertical bar represents the execution time of a single transaction. The vertical bars that cross the horizontal line represent transactions that exceeded the deadline.

To further understand the behavior of the timing failure detection mechanism we have executed the benchmark in the presence of an additional load that stresses the network and the server machines. This way, we have executed the real-time performance benchmark and, in random moments, we have executed the additional workload during a random amount of time. The additional workload has been adopted from the TPC-C performance benchmark [14] (this workload has been chosen due to practical reasons and any other workload could have been selected). The average number of transactions executed per minute decreased about 40% and the average transactions execution time increased around 30%. The latency remained the same.

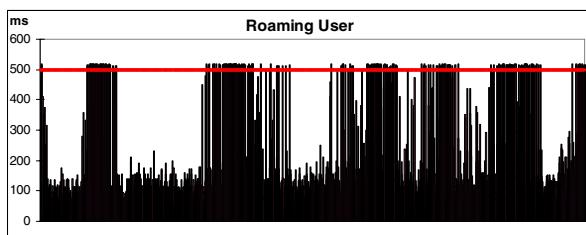


Fig. 5. Execution profile using timing failure in the presence of an additional database load

Figure 5 presents the execution profile for the roaming user transaction during one execution of the benchmark in the presence of the additional load. Note that, there are now many more transactions whose execution times exceed the deadline or gets closer to it. Nevertheless, all the timing failures were detected.

5 Conclusion

This paper discussed the problem of timing failure detection in database applications and proposes a transaction programming approach to help developers in programming database applications with time constraints. Three classes of transactions are considered concerning temporal requirements: transactions with no temporal requirements (typical ACID transactions), transactions with strict temporal requirements, and transactions with relaxed temporal requirements. The approach proposed implements these classes of transactions by allowing concurrent detection of timing failures during transaction execution. Timing failure detection can be performed at the database clients' interface, in the database server, or in a distributed manner. The paper illustrates the proposed programming models in a practical example using the Oracle 10g DBMS. A performance benchmark for real-time database applications is used to validate the approach and to show the advantage of timing failure detection.

From the results presented in this paper it is clear that it is useful to consider a new transaction programming approach aimed at supporting timing specifications for the execution of transactions. On the other hand, the work done so far was instrumental to uncover some of the problems that must be addressed to solve the temporal issues related to timing failure detection in DBMS settings. We intend to pursue this work and redesign or complement the mechanisms provided by the TCB for timing failure detection, so they become better suited to support the several classes of timed transactions that we identified as the fundamental ones.

References

1. J. Gray and A. Reuter, "Transaction Processing: Concepts and Techniques", The Morgan Kaufmann Series in Data Management Systems, Jim Gray, 1993.
2. M. Vieira, A. Costa, H. Madeira, "TACID Transactions", IEEE/IFIP Intl Conference on Dependable Systems and Networks, 1st Workshop on Hot Topics in System Dependability (HotDep-05), Yokohama, Japan, June 2005.
3. K. Ramamritham, "Real-Time Databases", Intl Journal of Distributed and Parallel DBs, 1996.
4. G. Ijzsoyoilu, R. T. Snodgrass, "Temporal and Real-Time Databases: A Survey", IEEE Transactions On Knowledge and Data Engineering, 1995.
5. L. DiPippo, V. Wolfe, "Real-Time Databases", Database Systems Handbook, Multiscience Press, 1997.
6. SIGMOD Record, Special Section on Advances in Real-Time Database Systems, Vol 25, number 1, pp.3-40, 1996.
7. T. Chandra, S. Toueg, Unreliable Failure Detectors for Reliable Distributed Systems, Journal of the ACM, 43(2), 225–267, 1996.

8. L. Dwork, L. Stockmeyer, "Consensus in the Presence of Partial Synchrony", Journal of the ACM, 1988.
9. F. Cristian, C. Fetzer, "The Timed Asynchronous Distributed System Model", IEEE Transactions on Parallel and Distributed Systems, 1999.
10. P. Veríssimo, A. Casimiro, "The Timely Computing Base Model and Architecture", Trans. on Computers - Special Issue on Asynch. Real-Time Systems, 2002.
11. J. Lindström and T. Niklander, Benchmark for Real-time Database Systems for Telecom., VLDB 2001 Intl Workshop on DB in Telecom. II, Rome, Italy, 2001.
12. Oracle Corporation, "Oracle® Database Concepts 10g Release 1 (10.1)", 2003.
13. M. Vieira and H. Madeira, "Recovery and Performance Balance of a COTS DBMS in the Presence of Operator Faults", Intl Performance and Dependability Symposium (jointly organized with DSN-2002), IPDS2002, Bethesda, Maryland, USA, June 2002.
14. Transaction Processing Performance Council, "TPC Benchmark C, Standard Specification, Version 5.4", 2005, available at: <http://www.tpc.org/tpcc/>.

BioDIFF: An Effective Fast Change Detection Algorithm for Biological Annotations

Yang Song¹, Sourav S. Bhowmick^{1,2}, and C. Forbes Dewey Jr.³

¹ School of Computer Engineering, Nanyang Technological University, Singapore

² Singapore-MIT Alliance, Nanyang Technological University, Singapore

³ Division of Biological Engineering, Massachusetts Institute of Technology, USA
assourav@ntu.edu.sg, cfdewey@mit.edu

Abstract. Warehousing heterogeneous, dynamic biological data is a key technique for biological data integration as it greatly improves performance. However, it requires complex maintenance procedures to update the warehouse in light of the changes to the sources. Consequently, a key issue to address is how to detect changes to the underlying biological data sources. In this paper, we present an algorithm called BioDIFF for detecting exact changes to biological annotations. In our approach we transform heterogeneous biological data to XML format and then detect changes between two versions of XML representation of biological data. Our algorithm extends X-Diff, a published XML change detection algorithm. X-Diff, being designed for any type of XML data, does not exploit the semantics of biological data to reduce the data set of bipartite mapping. We have implemented BioDIFF in Java. We have conducted an extensive performance study using data from EMBL, GenBank, SwissProt and PDB. Our experimental results show that BioDIFF runs 1.5 to 6 times faster than X-Diff.

1 Introduction

Technological advances in high throughput screening coupled with the genomic revolution resulted in a large amount of life sciences data that are often stored in geographically distributed databases. Some of the key features of these databases are as follows [6]. (1) Many data sources are typically centered on one *primary* class of objects, such as genes, proteins, or DNA sequences. (2) The primary objects are furthered described by a set of nested fields, called *annotations*. Many of the annotations are text fields, such as description, functional annotation, source of biomaterial etc. (3) Databases heavily cross-reference each other. (4) Databases overlap in the objects they represent, storing sometimes redundant and sometimes conflicting data. As valuable information is scattered around over literally hundreds of these databases, a data integration system that can handle heterogeneous, complex, and geographically dispersed biological data sources is a key area of research in bioinformatics [8].

Biological data integration approaches can be broadly categorized into three types. The first approach provides a uniform Web interface to various databases and analysis tools [2]. These systems usually use CGI scripts or Java servlets to execute queries against databases, to call analysis programs, or to search file-based data repositories. The second approach focuses on formulating complex declarative queries that span

The figure consists of two parts. Part (a) shows the EMBL Sequence Version Archive interface with a search bar for 'Accession Number or Sequence Version' containing 'AY278488'. It displays 2 matches: one for version 2 (AY278488.2) and one for version 1 (AY278488.1). Both entries have an issue date of '02-MAY-2003'. Part (b) shows a 'Differences' view for AY278488 between '22-APR-2003' and '02-MAY-2003'. The differences are listed in three columns: 'Lines unchanged' (green), 'Lines removed' (orange), and 'Lines inserted' (yellow). The inserted lines show the addition of new information such as 'SARS coronavirus BJ01, complete genome.' and 'SARS coronavirus BJ01, partial genome.'

(a) EMBL interface.

(b) Change detection.

Fig. 1. Change detection in EMBL

multiple heterogenous databases [3]. These two approaches have several disadvantages. First, efficiency of the application may be choked by the slowest external data source or by communication latency in the execution of queries and programs. Second, we may not be able to run our applications at a time we wish because a needed external source is unavailable. Third, there is always the risk of unintended “denial of service” attacks on the original sources. Finally, as many biological sources have large number of errors, there is always the risk of running remote applications that are sensitive to certain errors that cannot be detected nor corrected on-the-fly.

The third approach addresses the above limitations by taking a *warehousing* approach [1][3][7]. The first step in this approach is to develop a unified data model that can accommodate all the information that is contained in the various source databases. The next step is to develop a mechanism that will fetch the data from the source databases, transform them to match the unified data model and then load them into the warehouse. The warehouse then can be used for answering any of the questions that the source databases can handle, as well as those that require integrated knowledge that the individual sources do not have.

The warehouse approach greatly improves performance [3]. However, as biological data is highly dynamic, it requires complex maintenance procedures to update the warehouse in light of the changes to the sources. This raises a number of practical problems [3]: (1) How can we detect that the underlying data sources have changed and what are these changes? (2) How can we automate the refresh process? (3) How can we track the origins or “provenance” of data? *In this paper, we focus on the first issue.*

An important aspect of any change detection problem is finding out exactly how the underlying data source has changed. In the case of biological databases, this is complicated by the fact that updates are typically propagated in one of the three ways [3][5]: (1) Producing periodic new versions that can be downloaded by the user community (2) Timestamping data entries so that users can infer what changes have occurred since

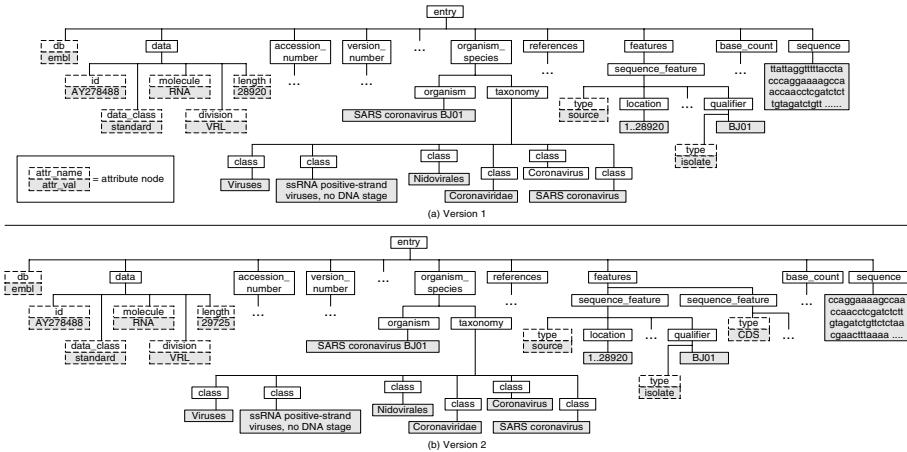


Fig. 2. XML representation of biological data (partial)

they last accessed the data (3) Keeping a list of additions and corrections; each element of this list is a complete entry. The list of additions can be downloaded by the user community. However, to the best of our knowledge, none of these methods precisely describe the minimal changes that have been made to the data. We illustrate this with an example.

Suppose that a warehouse stores a portion of EMBL data including data related to SARS (Severe Acute Respiratory Syndrome) virus. The EMBL data bank timestamps the data entries (Figure 2(a)) so that the warehouse maintainer can infer the latest version of the entry. The web site also provides a tool to compare the differences between two versions of SARS data by clicking the “Compare Selected” button in Figure 2(a). The differencing tool then highlights the changes by color coding the lines that are inserted, deleted, or remained unchanged during the transition as shown in Figure 2(b). The main drawback of this tool is that it does not exactly say how the entry has changed. The actual change may be very small. For example, consider the `ID` attribute in EMBL data. The general format of an `ID` line in EMBL is: `ID entryname dataclass; molecule; division; sequence length BP`. From Figure 2(b) it is clear that only the sequence length is modified from “28920” to “29725”. Values of the remaining attributes of the `ID` line are unchanged. However, the differencing tool in Figure 2(a) does not try to identify the exact change in the `ID` line. Rather, it represents the change as deletion and insertion of the `ID` line. Observe that the differencing tool represents an update of a line as a combination of deletion of the line followed by insertion of a new line (first two lines in Figure 2(b)). Assuming that the warehouse uses a relational database to store data, finding the exact change is important as it reduces the number of tables or tuples needed to be updated in the warehouse [3].

In this paper, we present an algorithm BioDIFF that can identify *exact* changes to the *annotations* associated with primary biological objects¹. In the rest of the paper,

¹ A preliminary and shorter version of this paper appeared as a poster in [9].

we use the genomic and proteomic data sources as running examples to illustrate our change detection technique. However, as we shall see later, our approach is generic and can be applied to any biological annotations. Note that we do not discuss detecting changes to primary objects (e.g., protein sequences, nucleotide sequences) as the differences between two primary objects (say nucleotide sequences) can be computed using a modified sequence comparison algorithm instead of the *matching algorithm* used in BIODIFF for annotations. As there is a significant body of work on sequence comparison techniques, we do not focus our discussion on detecting changes to primary objects here.

In our approach, we first transform data (e.g., flat files) from various biological data sources to XML format using Bio2X [10]. Then, we can address the problem of detecting changes to biological data in the context of such XML documents. Consequently, BIODIFF takes as input two versions of XML representation of biological data and compute the changes. Since there are several recent efforts in the XML research community to develop change detection algorithm for XML documents [2][11], an obvious issue is the justification for designing a separate algorithm for detecting changes to the XML representation of biological data. In fact according to [3][5], XML change detection algorithms can be directly used to detect changes to XML representation of biological data. We argue that although such algorithms will clearly work for annotation data, they are not efficient as they do not exploit the *semantics* of biological data. For instance, the min-cost max-flow algorithm for computing the bipartite mapping between two XML trees is the most time consuming part in X-Diff [1]. Hence, it is desirable to reduce the size of data set during mapping. However, X-Diff fails to do so for biological data as it ignores the semantics of the XML elements. BIODIFF is developed to address this issue by extending X-Diff[1]. As we shall see later, it exploits the semantics of the XML elements to further reduce the data size for bipartite mapping. Consequently, our experimental results (Section 3) show that BIODIFF runs 1.5 to 6 times faster than X-Diff on genomic and proteomic annotations.

2 Algorithm

We have designed and implemented a wrapper called Bio2X that converts flat files into hierarchical XML form based on extraction rules[10]. We have observed that XML representation of data from many major biological sources can be considered as unordered. For example, consider the XML tree representation of two versions of EMBL data in Figure 2. Assume that the nodes `<class>Coronavirus</class>` and `<class>Nidovirales</class>` swap their positions in Figure 2. However, this change is not significant since the order does not influence the semantics of the biological data entry. Hence, in this paper we assume that an unordered XML model is more appropriate for representing biological data.

The pseudocode of BIODIFF algorithm is given in Figure 3(a) and can be best described by the following five phases: the *identifier checking phase*, the *type classification phase*, the *parsing and hashing phase*, the *matching phase*, and the *edit script generation phase*. The *identifier checking phase* takes as input two XML documents,

<pre> Input: XML documents D_1 and D_2, DTD K Output: Edit script E /* Identifier checking phase */ (1) Set $id1$=identifier of D_1; (2) Set $id2$=identifier of D_2; (3) if ($id1=id2$) return E_{no_change}; /* Type classification phase */ (4) if (TypeContainer C is empty) $C = ChooseType(K)$; /* Parsing and hashing phase */ (5) parse D_1 to XTree tree1; (6) XHash(T_1); (7) parse D_2 to XTree tree2; (8) XHash(T_2); /* Matching & edit script generation phase */ (9) for each node $model$ in first-level elements of tree1 { (10) set name=tree1.GetName(node); (11) set node2=tree2.GetNode(name); (12) if (node2 == null) (13) add Delete(model, tree1) to E; (14) else { (15) if type1(model) (16) E=OneToOne(model, node2, E); /* Figure 3(c) */ (17) else if type2(model) (18) E=IdentElemComp(model, node2, E); /* Figure 4(a) */ (19) else if type3(model) (20) E=IdentSignature(model, node2, E); /* Figure 4(b) */ (21) else E=bipartite(model, node2, E); (22) } (23) } (24) for each node2 in tree2 but not in tree1 (25) add Insert(node2, tree2) to E; (26) return E; </pre>	<pre> Input: DTD K rooted at E Output: TypeContainer C (1) if no sub-element of E (2) $C \leftarrow \{type_1\}$; (3) for each sub-elements S of E { (4) add ChooseType(S) to C; (5) store <name, attributes> pair of S into M; (6) } (7) if M is not empty { (8) if each has distinct name (9) add type_1 into C; (10) else { (11) if none has any attributes (12) add type_2 into C; (13) else if all has the same attributes (14) add type_2 into C; (15) else { (16) if the attribute is not an id (17) add type_3 into C; (18) else (19) add type_4 into C; (20) } (21) } (22) } (23) return C; </pre>
--	--

(a) Main algorithm.

(b) Algorithm *ChooseType*.

Fig. 3. BioDIFF algorithm

new and old versions (denoted as D_1 and D_2), and determines whether they are identical. The equality of two XML representation of biological data can be concluded without parsing the entire XML documents. For instance, for genomic data sources, each biological data record has a *version identifier*. Whenever the data is changed, a new identifier will be assigned. So the identifiers of two data files can be extracted and compared first to determine whether the data files are identical. Similar identifiers can be identified for most of the important biological data sources. If the identifier checking phase detects that the two entries are not identical, then BioDIFF will parse the schema information of the documents (DTD/XML Schema) to classify the XML elements into four *types* depending on their structure. Such type classification information shall be used in the matching phase to minimize bipartite matching of the dataset. Note that if the DTD information is not available then it can be automatically generated from the XML documents using XTRACT[4]. We shall elaborate on this phase further later. Next, D_1 and D_2 are parsed into DOM Trees $tree1$ and $tree2$ in the *parsing and hashing phase*. The steps are similar to X-Diff[11] except for one key difference. Unlike X-Diff, when we parse the XML file, we encode the elements in the XML document with appropriate *type* of matching techniques based on the *type* information generated from the preceding phase. Note that if D_1 and D_2 contain primary objects (protein or nucleotide sequence) then they are excluded from parsing into nodes in the DOM trees. The goal of the *matching phase* is to compute the minimum cost matching between the DOM trees $tree1$ and $tree2$. We elaborate on this step later. In the *edit script generation phase*, we generate a minimum-cost edit script for changes to annotation data based on the minimum cost matching found in the matching phase. This step is similar to the one in X-Diff.

```

Input: XTree tree1, XTree tree2, edit script E
Output: the edit script E

(1) if hashcode(tree1)==hashcode(tree2) return E;
(2) store the signatures of first-level elements of
   tree1 in HashMap1;
(3) store the signatures of first-level elements of
   tree2 in HashMap2;
(4) for each element in HashMap1 {
(5)   if (existing mapping element in HashMap2) {
(6)     if (it is an element node) {
(7)       if type1(node1)
          E=OneToOne(node1, node2, E);
        else if type2(node1)
          E=IdenElemComp(node1, node2, E);
      else if type3(node1)
          E=IdenSignature(node1, node2, E);
        else
          E=bipartite(node1, node2, E);
    }
  }
(12) else {
(13)   compare their values;
(14)   if different
(15)     add Update(element1, element2) to E;
(16)   }
(17) }
(18) else
(19)   add Delete(element, tree1) to E;
(20) }
(21) for each element in HashMap2 {
(22)   if (no mapping element in HashMap1)
(23)     add Insert(element, tree2) to E;
(24) }
(25) return E;

```

(a) Function *OneToOne*.

```

Input: XTree tree1, XTree tree2, edit script E
Output: the edit script E

(1) if hashcode(tree1)==hashcode(tree2) return E;
(2) store the values of sub-elements of tree1
   in Vector1;
(3) store the values of sub-elements of tree2
   in Vector2;
(4) for each element value in Vector1 {
(5)   if (contained in Vector2)
(6)     mark unchanged;
(7)   }
(8) for each element value in Vector2 {
(9)   if (changed) {
(10)     choose the next changed element in Vector1;
(11)     if (exist)
(12)       add Update(element1, element2) to E;
(13)     else
(14)       add Insert(element2, tree2) E;
(15)   }
(16) }
(17) for each element leaf in Vector1
(18)   add Delete(element1, tree1) to E;
(19) return E;

```

(b) Function *IdenElemComp*.

Fig. 4. BIODIFF algorithm (contd.)

2.1 Type Classification Phase

The min-cost max-flow algorithm for computing the bipartite mapping between two XML trees is the most time consuming part in X-Diff. Hence, a key goal of BIODIFF is to minimize bipartite mapping computation by exploiting the semantic relationship between various nodes in the XML tree. For example, the *data* element in Figure 2 contains only attribute values, whereas the *organism-species* element contains a list of subtree elements. Such differences in the structure of the subtrees are exploited in our approach to achieve this goal.

In this phase, we classify the XML elements into four different *types* based on their structures (we shall elaborate on these types later) by analyzing the DTD (or XML schema). As we shall see in the matching phase, instead of applying expensive bipartite matching for all cases, we apply four different matching techniques to the XML elements based on the types they belong to. Three of these matching techniques run in linear time in contrast to polynomial time complexity of bipartite matching.

The algorithm to classify the elements in the DTD to different types is shown in Figure 3(b). It takes as input the DTD of the XML representation of biological data and returns as output the *TypeContainer* *C* which contains information about different XML elements and corresponding types. The *ChooseType* function is invoked for each element in the DTD recursively and at each level, the names and attributes of all the subelements are examined to choose the type of the current element. Let us illustrate this with a simple example. Consider the subtree structure rooted at *organism-species* in Figure 2. *ChooseType* is first invoked for its subelement *organism* in the DTD, which is determined to be of *Type 1* since it has no subelements. *ChooseType* is then invoked for *taxonomy* and it is classified as *Type 2* since it contains a list of subelements having identical names (*class*). Finally, as the *organism-species* contains two distinct subelements, *Type 1* is chosen for this element.

```

Input: XTree tree1, XTree tree2, edit script E
Output: the edit script E

(1) set attrname=the names of key attributes;
(2) for each first-level elements in tree1 and tree2
    include name of attrname into signature;
(3) t1=subtree containing nodes with distinct
    signatures in tree1;
(4) t2=subtree containing nodes with distinct
    signatures in tree2;
(5) E=OneToOne(t1, t2, E);
(6) t1=subtree containing nodes without attributes
    in tree1;
(7) t2=subtree containing nodes with attributes
    in tree2;
(8) E=IdenElemComp(t1, t2, E);
(9) for each signature common to multiple elements {
(10)   t1c1=subtree containing nodes with this
        signatures in tree1;
(11)   t2c2=subtree containing nodes with this
        signatures in tree2;
(12)   E=IdenElemComp(t1c1, t2c2, E);
(13) }
(14) return E;

```

```

<?xml version="1.0" encoding="UTF-8"?>
.....
<coordinate section>
<atom serial no="1" name="N" residue name="VAL"
chain id="A" seq num="1" x-coord="-38.199"
y-coord="-40.257" z-coord="97.510" occupancy="1.00"
temp factor="22.53" />
<atom serial no="2" name="CA" residue name="VAL"
chain id="A" seq num="1" x-coord="-38.816"
y-coord="-41.316" z-coord="96.669" occupancy="1.00"
temp factor="18.15" />
</coordinate section>
.....

```

(a) Function *IdenSignature*.

(b) Element coordinate_section of PDB.

Fig. 5. BioDIFF algorithm (contd.)

2.2 Matching Phase

We now discuss the matching process in BioDIFF. Unless specified otherwise, in our following discussion we use the notion of signature as introduced by [11]. Observe that the first level element nodes (elements for brevity) in the tree representation of XML version of biological data have *distinct* structures (Figure 2). Each node has a unique name and hierarchy. Each node in the first level appears only once and mapping occurs only between nodes with the same signature. So the whole XML tree can be divided into a set of smaller subtrees rooted at each first-level node. Each smaller tree will be compared with another smaller tree from the second XML tree having the node with same name. For example, the subtree rooted at node labeled *organism-species* in Figure 2(i) will be compared with the subtree in Figure 2(ii) whose root is in the first level and has the same label. Note that this computation is independent from the remaining subtrees.

The above step alone does not provide performance improvement compared to X-Diff. X-Diff also achieves this with its signature definition. However, this step makes it possible to use *different* types of matching for *different* subtrees. Hence, we categorize the matching techniques into four basic types for both minimum-cost distance computation and minimum cost edit script generation. We discuss these techniques in detail now. Note that each subtree is treated independently for further matching. So any distance computation will be localized within each subtree.

Type 1: One-to-One Comparison. An element in the XML representation of biological data can be composed of a number of attributes, subelements or textual data. Some elements may exhibit unique signature within its subtree scope. For example, consider the *organism* element in Figure 2. Only one *organism* element can exist in the subtree rooted at *organism-species* element. So there will be at most one candidate-matching element in the newer version of the subtree. Hence, one-to-one comparison can be conducted between these two elements. Lines 1-2 in Figure 3(b) identifies these types while parsing the DTD.

Case 1: The element contains only text value: The corresponding element pair from the older and newer versions can be matched directly using one-to-one comparison. For example, the *version_number* element shown in Figure 2 contains only a string value representing the accession number and version number. So the two *version_number* elements in Figure 2 can be compared directly.

Case 2: The element contains distinct attributes: Each attribute should have a distinct name (signature). Each attribute can be matched with only a single attribute of the other XML file. The entire attribute list pair is then matched on a one-to-one basis. Take the *data* element in Figure 2(i) as an example. It contains the attributes *id*, *data_class*, *molecule*, *division*, and *length*. After obtaining the first attribute *id* in Figure 2(b)(i), the algorithm searches for attribute *id* in Figure 2(ii) in the element node *data* and compares their content. This is repeated for each attribute of *data*.

Case 3: The element contains a list of subelements with distinct names: One-to-one comparison is also sufficient to match the two lists. For example, consider the *dates* element in Figures 2(i) and 2(ii). Each *dates* element consists of two subelements: *created* and *updated*. These two subelements will appear at most once in this subtree rooted at *dates*. So given an element *created* in the older version of the subtree, there will be at the most one candidate-matching element in the newer version of the subtree. After matching the two *created* nodes, for example, further matching between the two subtrees of *created* is then computed recursively. The type of matching technique to be applied on the subelements depends on the specific structure again.

The pseudocode for one-to-one comparison is shown in Figure 4(a). It can be seen that with the utilization of a *HashMap* structure for storing elements, linear time complexity $O(|T_1| + |T_2|)$ is achieved, where $|T_1|$ and $|T_2|$ are the numbers of nodes in the subtrees T_1 and T_2 respectively. Note that although X-Diff only performs min-cost max-flow computation between the elements with the same signature, for a subtree with all unique signature subelements, the complexity will not reduce to $O(|T_1| + |T_2|)$. It still needs to enumerate all the pairs first to select the nodes with the same signature as it cannot know ahead of time that only one matching candidate actually exists.

Type 2: Identical Subelement Comparison. The first case discussed above is suitable for an element with a list of distinct subelements or attributes where each element or attribute must have a unique name within the scope of its subtree. Conversely, an element may contain a list of subelements having identical names. In this case, by following the X-Diff convention, if all the subelements have the same signature, then they will be compared using min-cost max-flow algorithm. However, as bipartite matching is an expensive procedure, we resolve this program by transforming a bipartite matching problem into linear-time matching whenever possible. Lines 10-14 in Figure 3(b) identifies the cases below while parsing the DTD.

Case 1: Elements without attributes: For example, the *taxonomy* element has a list of subelements with the same name *class*. Each of the *class* element does not contain any attribute. A *class* can be deleted, inserted, or updated. In our approach, we first find all the unchanged *class* pairs and then we assign all the unmatched *class* elements in the second subtree to unmatched elements in the first subtree sequentially and record

that they are changed. If a *class* element of the second subtree is left unpaired, then it is considered as newly inserted. Similarly, if a *class* element of the first subtree is left unpaired, then it is considered as deleted.

Case 2: Elements having identical attributes: Each element with identical name may have identical nonempty set of attribute name-value pairs. Consequently, this becomes identical to Case 1 where the elements are just identical by examining the titles of the nodes. Hence the same comparison technique will be applied to them.

Case 3: Elements of cases 1 and 2 containing subtree structures: The sequential matching becomes no longer valid since the matching pairs have to be computed by analyzing the entire hierarchy of the subtrees. A min-cost max-flow bipartite matching has to be carried out. However, such structure does not exist in our XML data formats generated by Bio2X [10]. Any element, which is not unique within the current scope and contains a subtree structure, will have at least one distinct attribute associated with it. Hence, such situation can be ignored.

The algorithm is shown in Figure 4(b). The complexity of this procedure is also linear in $O(|T_1| + |T_2|)$.

Type 3: Extended Signature Comparison. We now consider the case when identical subelements have different attributes (name or value). They can be text-value, attribute-value elements, or contain subelements. These two cases are defined by the Lines 16-17 in Figure 3(b).

Case 1: Each identical-name element has a distinct attribute: We can differentiate these elements by utilizing the attributes as identifiers. By incorporating the attribute name and value into the original signature, we generate unique signatures for each element within the current subtree. Hence, the matching problem will be reduced to a one-to-one comparison again as described earlier. For instance, the *qualifier* elements (Figure 2(i)) has a signature *entry/features/sequence_feature/qualifier/element* as defined by X-Diff. Observe that three of them have an attribute labeled *type*. Hence they can have the following distinguishable signatures: *entry/features/sequence_feature/qualifier/element/mol_type*, *entry/features/sequence_feature/qualifier/element/isolate*, and *entry/features/sequence_feature/qualifier/element/organism*. Consequently, the problem of matching an element in two versions of XML documents transforms into locating an element with the same refined signature. This is basically the one-to-one comparison algorithm described earlier. One *qualifier* element in Figure 2(i) has no *type* attribute. It will be matched with the other *qualifier* element without *type* (Figure 2(ii)). Multiple *qualifier* elements without attributes can exist in one subtree. Then they will be extracted and matched using the Type 2 technique. Also, if multiple *qualifier* elements have identical attributes, then they are also matched using Type 2 technique.

Case 2: Each identical-name element has multiple different attributes: The technique stated above is easily extensible to an element having multiple attributes. In this case, either any one of the attribute acts as the identifier or some of the attributes can be combined to act as the identifier. The attribute names are predefined for each XML element applying this matching algorithm. For example, consider the element *<database_reference*

`db="EMBL" primary_id="L09112" secondary_id="AAA96323.1" />`, which has three difference attributes. The value of attribute `primary_id` or `secondary_id` is always unique within a database. Hence, attribute `primary_id` with `db` can be used as the identifier for this element. We do not choose `secondary_id` because it is an optional attribute. The choice of the key attribute or a combination of attributes depends on the biological context of the data, which is predefined in the biological databases.

The complete matching algorithm is outlined in Figure 5(a). After extending the signature, the elements will be matched using either Type 1 or 2 technique. Note that the complexity of the for loop (line (9) to (13)) is linear to number of nodes matched by Type 2 technique, which are subsets of T_1 and T_2 . Hence, this matching technique also runs in linear time complexity: $O(|T_1| + |T_2|)$.

Type 4: Bipartite matching. The min-cost max-flow algorithm as used in *X-Diff* is used for elements that cannot be matched using the above three methods or a combination of them. This may occur when the attribute(s) is only an index for numbering the elements and the element pair with the same index may not match at all. For example, the `coordinate_section` of PDB database contains a list of `atom` elements (Figure 5(b)). The atom list records the names, locations and the atomic coordinates for standard residues. Each `atom` is identified by a `serial_number`, and the sequence of `atom` elements is determined by the order of the corresponding residue. Suppose in the newer version, an extra atom element is added for a newly inserted residue. The `atom` may be inserted in the middle of the atom list. When we compare the old and new version of `atom` list, the `atom` newly inserted will be matched to some `atom` in the old version, which is not correct. Hence, the alignment between the `atom` list of the old and new versions cannot be carried out reliably by matching the `atom` elements with the same `serial_number` (signature). Consequently, the only option left is to use min-cost max-flow bipartite matching. Note that this procedure requires $O(|T_1| \times |T_2| \times \max\{\deg(T_1), \deg(T_2)\} \times \log_2(\max\{\deg(T_1), \deg(T_2)\}))$ complexity [11].

2.3 Complexity Analysis

Let $|T_1|$ and $|T_2|$ be the numbers of nodes in the two XML trees T_1 and T_2 respectively. Let n_1 and n_2 be the numbers of nodes requiring a min-cost max-flow matching algorithm. In the equality checking phase, the equality of two documents is determined by checking their identifier values. The time complexity to locate identifiers from two documents is $O(L_1 + L_2)$, where L_1 and L_2 are the numbers of nodes ahead of the identifiers. In the parsing and hashing phase, the time complexity to parse two documents and construct trees is $O(|T_1| + |T_2|)$. Hashing is performed during parsing. Since we need to sort child node XHash values before computing parent node XHash values, the upper bound of the complexity is $O(|T_1| \times \log |T_1| + |T_2| \times \log |T_2|)$ [11]. In the matching phase different matching algorithm is applied to each subtree rooted at first-level node according to its node type. For Type 4 matching the time complexity is $O(n_1 \times n_2 \times \max\{\deg(n_1), \deg(n_2)\} \times \log_2(\max\{\deg(n_1), \deg(n_2)\}))$. The time complexity for Type 1 to 3 matching techniques is $O(|T_1| + |T_2|)$. So the overall time complexity of the matching phase is $O(|T_1| - n_1 + |T_2| - n_2 + n_1 \times n_2 \times \max\{\deg(n_1), \deg(n_2)\} \times \log_2(\max\{\deg(n_1), \deg(n_2)\}))$. It shows that the performance

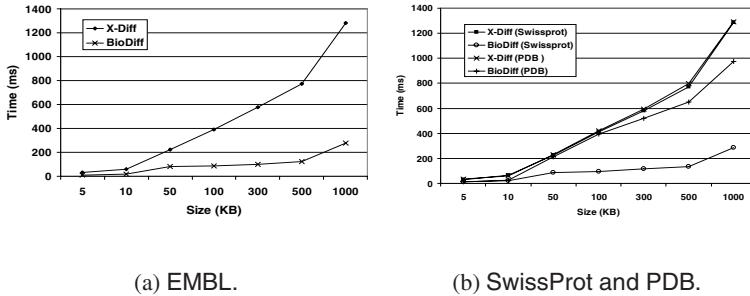


Fig. 6. Performance study using genomic and proteomic data

primarily depends on the number of nodes in the files and the percentage of nodes that require Type 4 matching. When the changes between the two versions are not reflected in $|T_2|$ or n_2 , the percentage of changes does not affect the performance as well, which is consistent with the experiments conducted in X-Diff[11]. Finally, for the minimum cost edit script generation phase, the minimum cost edit script is generated by traversing all nodes in the two trees. Hence, the complexity is $O(|T_1| + |T_2|)$.

Let us compare the time complexity of the matching process for nonsequence data in BioDIFF with X-Diff. As X-Diff requires about $O(|T_1| \times |T_2| \times \max\{\deg(T_1), \deg(T_2)\} \times \log_2(\max\{\deg(T_1), \deg(T_2)\}))$ to match, the improvement of nonsequence element matching in BioDIFF over X-Diff can be estimated as $O(|T_1| \times |T_2| \times \max\{\deg(T_1), \deg(T_2)\} \times \log_2(\max\{\deg(T_1), \deg(T_2)\}) / (|T_1| + |T_2| - n_1 - n_2 + n_1 \times n_2 \times \max\{\deg(n_1), \deg(n_2)\} \times \log_2(\max\{\deg(n_1), \deg(n_2)\})))$. Assuming $n_1 = n_2 = n$, $|T_1| = |T_2| = t$, and $\max\{\deg(T_1), \deg(T_2)\} \times \log_2(\max\{\deg(T_1), \deg(T_2)\}) = x$, the complexity comparison becomes $O(xt^2 / (t - n + xn^2))$. For certain value of t , this percentage of speed up depends on the value of n . If $n = 0$ or $n = 1$, then the speed up is around $O(xt)$, which is the best-case performance. If $n = t$, then the speed up is equal to 1, which is the worst-case performance. If $n > 1$ and $n < t$, then the speed up is between 1 and $O(xt)$. In conclusion, the change detection performance of BioDIFF algorithm on annotations is always faster than X-Diff. However, the speed up depends on the percentage of elements utilizing the linear time matching methods.

3 Performance Study

BioDIFF is implemented in Java using J2SDK 1.4.0. Particularly, we investigate the impact of file size on response time of BioDIFF and X-Diff. We ran the experiments on a Pentium III 900MHz PC with 256 MB memory under MS Windows 2000 Professional. X-Diff is downloaded from <http://www.cs.wisc.edu/~yuanwan/xdiff.html>.

The testing is done for GenBank, EMBL, Swiss-Prot, and PDB data separately. The size of the documents excluding the primary object (sequence data) ranges from 5 KB to 1 MB. As sequences occupy major chunk of space in the documents, we believe that 1 MB is large enough to represent annotations associated with primary biological objects.

Note that the execution time of X-Diff and BIODIFF algorithms only consists of the change detection time between the two XML trees. The preprocessing time, including the type generation and parsing of XML documents is not included in the performance evaluation. Note that preprocessing time depends only on the document and schema size and is not influenced by the algorithm efficiency. According to the complexity analysis, the proportion of nodes requiring Type 4 matching also affects the performance of matching phase. Hence, we choose data files with different proportion of tree structures that need Type 4 matching technique for each file size. The performance recorded is averaged from the testing results for each file size. For example, when testing BIODIFF on EMBL, Genbank, and SwissProt data, we took 10 files for each sample file size, with *references* elements occupying 1-10% of the entire file size. For PDB, we assumed 30-80% elements involved in Type 4 matching. Note that percentage of elements involved in Type 4 matching is chosen based on real life examples.

Figure 6 shows the results. We do not show the results on Genbank separately as its performance is similar to EMBL. It can be seen that BIODIFF outperforms X-Diff for all the four databases. As seen from the figures, X-Diff exhibits similar performance for different databases as it is a generic algorithm for all types of XML documents. Hence, its performance mainly depends on the number of nodes. BIODIFF, on the other hand, has different performance for each database since each database has different tree structures. If a database has more nodes that require min-cost max-flow matching algorithm, the improvement of BioDiff compared to X-Diff is less. For example, for Genbank, EMBL, and SwissProt, only the *references* element requires bipartite matching. Consequently, BIODIFF outperforms X-Diff 2 to 3 times for documents of size less than 100 KB. This increases to 6 times as the size of the documents increase to 1 MB. For PDB, *reference*, *coordinate_section*, and *secondary_structure* elements require bipartite matching. As number of nodes that requires min-cost max-flow matching algorithm for change detection is larger in PDB compared to Genbank, EMBL, and SwissProt, the execution time of BIODIFF on PDB is larger than the time taken for Genbank, EMBL, and SwissProt. Even then BIODIFF is almost 1.5 times faster than X-Diff for PDB dataset.

4 Conclusions

In this paper, we present an algorithm called BIODIFF for detecting exact changes to biological annotations. In our approach we transform heterogeneous biological data to XML format using Bio2X and then detect changes between two versions of XML representation of biological annotations. Our algorithm extends X-Diff [11], a published change detection algorithm for unordered XML. The min-cost max-flow algorithm for computing the bipartite mapping between two XML trees is the most time consuming part in X-Diff. BIODIFF addresses this limitation by exploiting the semantic relationship between various nodes in a subtree, attribute usage, presence or absence of optional elements, etc. Our experimental results show that BIODIFF runs 1.5 to 6 times faster than X-Diff.

References

1. Bahl,A. (2002) PlasmoDB: the Plasmodium genome resource. An integrated database that provides tools for accessing, analysing and mapping expression and sequence data (both finished and unfinished), *Nucleic Acids Res.*, **30**, 87-90.
2. Cobena,G., Abiteboul,S., Marian,A. (2002) Detecting Changes in XML Documents, *In Proc. of ICDE*, 41-52.
3. Davidson,S.,B., Crabtree,J., Brunk,B., *et al.* (2001) K2/Kleisli and GUS: Experiments in integrated Access to Genomic Data Sources, *IBM Systems Journal*, **40(2)**, 512-531.
4. Garofalakis,M.,N., Gionis,A., Rastogi,R., Seshadri,S., Shim,K.(2000) XTRACT: A System for Extracting Document Type Descriptors from XML Documents, *In Proc. of SIGMOD*, 165-176.
5. Hammer,J., Schneider,M. (2003) Genomics Algebra: A New, Integrating Data Model, Language, and Tool for Processing and Querying Genomic Information, *In Proc. of Conference on Innovative Data Systems Research (CIDR)*.
6. Leser,U., Naumann,F. (2005) (Almost) Hands-Off Information Integration for the Life Sciences, *In Proc. of CIDR*, 2005.
7. Ritter,O., Kocab,P., Senger,M., Wolf,D., Suhai,S. (1994) Prototype implementation of the integrated genomic database, *Comput. Biomed. Res.* **27**, 97115.
8. Stein,L.,D., (2003) Integrating Biological Databases, *Nature Rev Genet*, **4(5)**, 337-345.
9. Song,Y., Bhowmick,S.,S., (2004) BioDiff: An Effective Fast Change Detection Algorithm for Genomic and Proteomic Data. *In Proc. of ACM CIKM(Poster)*, 146-147.
10. Song,Y., Bhowmick,S.,S., (2005) Bio2X: A Rule-based Approach for Semi-automatic Transformation of Semistructured Biological Data to XML, *Data and Knowledge Engineering Journal*, **52(2)**, 249-271.
11. Wang,Y., DeWitt,D., Cai,J-Y, (2003) X-Diff: A Fast Change Detection Algorithm for XML Documents, *In Proc. of IEEE ICDE* , 519-530.
12. Zdobnov,E.,M., Lopez,R., Apweiler,R., Etzold,T., (2002) The EBI SRS server-recent Developments, *Bioinformatics*, **18(2)**, 368-373.

An Efficient Implementation for MOLAP Basic Data Structure and Its Evaluation

K.M. Azharul Hasan, Tatsuo Tsuji, and Ken Higuchi

Graduate School of Engineering, University of Fukui, Fukui-shi, 910-8507, Japan
`{hasan, tsuji, higuchi}@pear.fuis.fukui-u.ac.jp`

Abstract. In this paper we describe an efficient implementation scheme for MOLAP internal basic data structure based on extendible multidimensional arrays. In general, MOLAP implementation scheme employs multidimensional array as their basic data structure. But most of the cases the implemented arrays are very sparse when employed to store front end relational tables in OLTP systems. Moreover conventional multidimensional arrays cannot be extended when new column values needs to be added. In this paper, to solve these problems, the concept of extendible array is used. The effectiveness of extendible array for MOLAP implementation is shown by means of both theoretical analysis and experimental results.

Keywords: Data warehousing, MOLAP, Multidimensional Array, Extendible Array, OLAP Operation.

1 Introduction

Online analytical processing (OLAP) is becoming increasingly important for analyzing multidimensional data. This data is generally derived from transactional data using various levels of aggregation. This aggregation levels are maintained in data warehousing system. Basically there are two kinds of OLAP systems employed in data warehouses. One is for relational OLAP called ROLAP and the other is for multidimensional OLAP called MOLAP. The data cube operation proposed in [1] computes the group-by aggregations over all possible subsets of the specified dimensions. Much work has been done for computing ROLAP data cube [2]-[4] but few works on MOLAP data cube[5][6].

The MOLAP systems use a multidimensional data structure such as an array constructed from the original data, which are typically stored in relational databases. Conventional multidimensional arrays do not support dynamic extension of an array and hence addition of a new column value is impossible if the size of the dimension overflows. Therefore we need a method of extending multidimensional arrays in all dimensions. Another problem with the multidimensional array structure is its sparsity, which wastes memory because a large number of array cells are empty and thus are rarely used during the computation. In particular, the sparsity problem becomes serious when the number of dimensions increases. This is because the number of all possible combinations of dimension values exponentially increases, whereas the number of actual data values stored in a relational table would not increase at such a rate.

In this paper, we present a method to overcome the problems introducing a new data structure. The concept of *extendible array* [7] is employed in order to extend the multidimensional array dynamically. An extendible array is extendible in any direction without any relocation of the data already stored.

In our previous work [8], the History-Offset implementation of Relational Tables (HORT) based on the extendible array and its superiority over conventional implementation of relational tables is presented. Here in this paper we have reorganized our HORT data structure to be implemented for MOLAP implementation and show the effectiveness of our scheme by means of both theoretical analysis and experimental results.

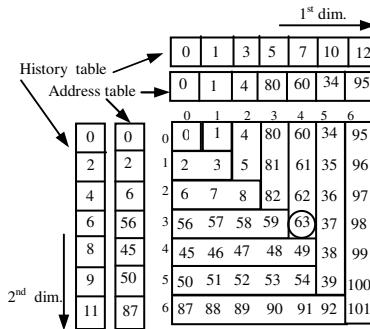


Fig. 1. Realization of 2 dimensional extendible array

2 Employing Extendible Arrays

An n dimensional extendible array A has a history counter h and three kinds of auxiliary table for each extendible dimension i ($i=1,\dots,n$). See Fig. 1. These tables are history table H_i , address table L_i , and coefficient table C_i . The history tables memorize extension history h . If the size of A is $[s_n, s_{n-1}, \dots, s_1]$ and the extended dimension is i , for an extension of A along dimension i , contiguous memory area that forms an $n-1$ dimensional subarray S of size $[s_n, s_{n-1}, \dots, s_{i+1}, s_{i-1}, \dots, s_2, s_1]$ is dynamically allocated. Then the current history counter value is incremented by one, and it is memorized on the history table H_i , also the first address of S is held on the address table L_i . An element $\langle i_n, i_{n-1}, \dots, i_1 \rangle$ in an n dimensional conventional fixed size array of size $[s_n, s_{n-1}, \dots, s_1]$ is allocated on memory using an addressing function like this:

$$f(i_n, i_{n-1}, \dots, i_2, i_1) = s_1 s_2 \dots s_{n-1} i_n + s_1 s_2 \dots s_{n-2} i_{n-1} + \dots + s_1 i_2 + i_1$$

Here, we call $\langle s_1 s_2 \dots s_{n-1}, s_1 s_2 \dots s_{n-2}, \dots, s_1 \rangle$ as a *coefficient vector*. Using these three kinds of auxiliary tables, the address of an array element can be computed as follows. Consider the element $\langle 4, 3 \rangle$ in Fig.1. Compare $H_1[4]=7$ and $H_2[3]=6$. Since $H_1[4] > H_2[3]$, it can be proved that the element $\langle 4, 3 \rangle$ is involved in the extended

subarray S occupying the address from 60 to 63. The first address of S is known to be 60, which is stored in $L_1[4]$. Since the offset of $\langle 4,3 \rangle$ from the first address of S is 3, the address of the element is determined as 63.

3 Implementing MOLAP by Extendible Array

3.1 The Data Structure

The data structure that is employed here is a two level tree structure. The first level of the structure is a one way list containing (key, pointer) pairs where key is the first key and pointer is the starting address of a node of the second level. The first level of the structure serves as a gateway to the second level. Hereafter the first level will be called as *head* and the second level will be called *leaf nodes* as shown in Fig. 2. The *leaf nodes* contain at most k keys.

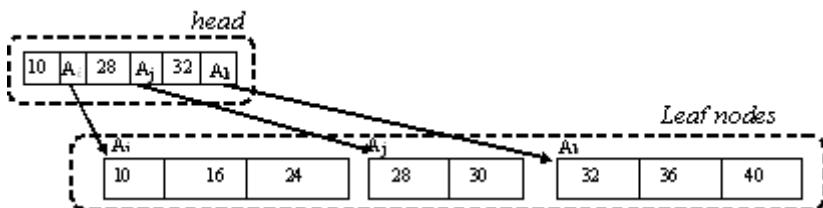


Fig. 2. The CDL data structure

When a key value K_i ($1 \leq i \leq k$) is to be inserted, the largest value smaller than or equal to K_i is determined in *head* and the pointer associated with the key value is followed to determine the corresponding *leaf node* of the second level. The key value K_i is inserted to the node. If the node overflows i.e. more than k keys needs to be entered in the node, the node is split into two in the middle such that one node contains $\lfloor k/2 \rfloor$ keys and another node contains $\lceil k/2 \rceil$ keys and the ascending order of the keys is maintained. The (key, pointer) pair of the new two node is stored into the *head*. The key K_i is then inserted to the appropriate leaf node. The (key, pointer) pair in the *head* are stored in such a way that *head* is ordered in terms of key values. Fig. 2 shows the data structure after inserting the keys 10, 16, 28, 32, 40, 24, 30, 36 in the order for $k=3$.

Whenever a key value K_i is searched, the largest value smaller than or equal to K_i is determined in the *head* and the pointer associated with the key value is followed to determine the corresponding leaf node which contains the desired key value K_i . To search a range of key values, the traversal is performed by determining the *leaf node* containing the lowest value in the range then sequential search is performed on the leaf nodes until the node containing highest value in the range is determined or the end of the node is reached. The next pointer in the *head* is followed to determine the next leaf node and searched the leaf node until the largest value in range is found. Binary search is adopted to search the *head*. The *head* is placed on main memory and

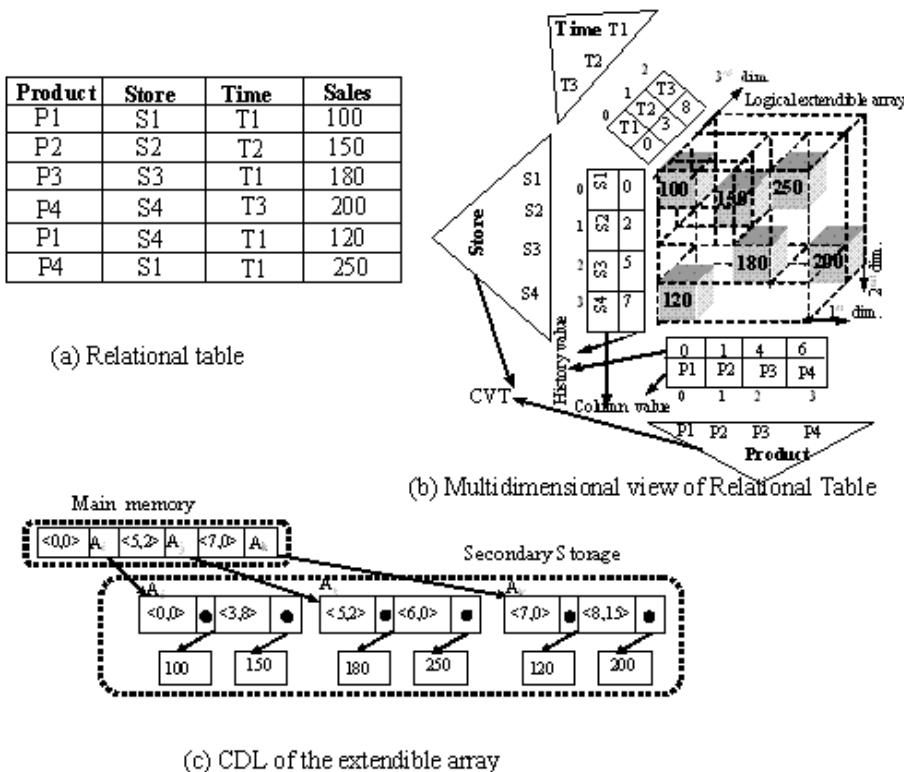


Fig. 3. Illustrative example of Compressing Extendible array

the *leaf nodes* are stored on secondary storage. In the following each of the key value K_i in leaf nodes will be associated with a data value D_i as shown in Fig. 3(c).

3.2 Compressing Sparse Array

It is desirable to develop data compression techniques so that the data can be accessed in their compressed form and operations can be performed using the compressed data. In our technique, we specify an element using the pair of *history value* and *offset value* of the extendible array. Since a history value is unique and has one to one correspondence with the corresponding subarray, the subarray including the specified element of an extendible array can be referred to uniquely by its corresponding history value h . Moreover the *offset value* (i.e., logical location) of the element in the subarray is also unique in the subarray. Therefore each element of an n dimensional extendible array can be referenced by specifying the pair (*history value*, *offset value*).

Consider a simple multidimensional model, in which we have the dimensions *product*, *store*, *time*, and the “measure” as shown in Fig. 3. There are three dimensions in Fig. 3 for product, store and time respectively. The sales values are stored in the corresponding cell of the extendible array as *fact data*. For example the sales

value 200 in the extendible array (Fig. 3(b)) indicates the fact that this is the sales value for product P4 of Store S4 in Time T3 corresponding to the coordinate <4,4,3>.

3.3 The Implementation Model

The model that we are going to present is based on the extendible array explained in Section 2. An example of physical implementation of the scheme is shown in Fig. 3.

Definition 1 (CVT). CVT_k for the k -th column of an n column relational table is defined as a structure of B⁺ tree with each distinct column value v as a key value and its associated data value is subscript i of the k -th dimension of the logical extendible array. Hence the entry of the sequence set of the B⁺ tree is the pair (v, i) . The reference of the subscript i includes history value and co-efficient vector of a subarray and the column value itself. The history value and coefficient vector are the auxiliary tables explained in Section 2.

Definition 2 (CDL). The set of the pairs (*history value*, *offset value*) for all of the effective elements in the extendible array are housed as the keys in a two level tree structure described in Section 3.1 called CDL (Compressed Data List). The corresponding fact data is inserted as its associated data value in the leaf nodes as shown in Fig. 3(c). We assume that the key occupies the fixed size storage and the *history value* is arranged in front of the *offset value*. Hence the keys are arranged in the order of the history values and keys that have the same history values are arranged consecutively in the leaf nodes of CDL.

The history value and offset value of a record is determined and stored in CDL having the fact data as the data value. A subarray is constructed for each distinct value of a column. The extension of an extendible array is performed logically and physically only the position information of the effective array elements is stored in CDL. Hence we will call the extendible array as *logical extendible array*.

4 OLAP Operations

The OLAP operations are the basis for answering questions like: “*find the sales value of product P4*” or “*find the sales value of store S1 in Time T1*” in relation Fig. 3(a). The former is *slice* operation and the later is *dice* operation. The slice is a selection along one dimension, while dice operation defines a sub cube by performing a selection on two or more dimensions. In the dice operation column values of some dimensions are specified. For example, in the predicate “*find the sales value of store S1 in Time T1*” the dimension *Store* and *Time* is specified having column values *S1* and *T1*. Let the specified column values be $v_{d_1}, v_{d_2}, \dots, v_{d_k}$ and their dimensions corresponds to d_1, d_2, \dots, d_k . Let $h_{d_1}, h_{d_2}, \dots, h_{d_k}$ be the history values that correspond to the column values $v_{d_1}, v_{d_2}, \dots, v_{d_k}$ and the maximum history value be $h_{\max} = \max(h_{d_1}, h_{d_2}, \dots, h_{d_k})$. The subarray corresponding to h_{\max} is named as the *principal subarray* in the following.

Only the principal subarray corresponding to h_{\max} is the candidate subarray for searching among the specified dimensions. The remaining candidate subarrays belong

to the unknown (i.e., column value not specified) dimensions and the history values of these subarrays are greater than h_{\max} .

Dice Operation

The operation starts finding the largest key value smaller than equal to the key $\langle h_{\max}, o \rangle$ in the *head* of CDL. After that, sequential search is performed to the rest of leaf nodes until the end of sequence set is reached; the sales values of the keys matching condition are added together to find the total sales value. Note that after searching the principal subarray, the key matching continues against the subarrays that have the history values greater than h_{\max} . But the subarrays that belong to the known (value-specified) dimensions do not include the candidate sales; they are read through without key matching.

The slice operation is similar to that of the dice operation because in the slice operation the number of known dimension is only one where as in dice operation the number of known dimension is more than one. Hence the slice operation can be performed as described above.

5 Cost Analysis

In this section, we model the processes of retrievals and extensions for MOLAP under two different implementation strategies namely Conventional Multidimensional Arrays (CMA) and Extendible Multidimensional Arrays (EMA). The first one reorganizes the array whenever there is an extension to it. That is, the whole array will be relinearized on disk to accommodate the new data due to the addition of new column values. The second strategy extends the initial array with subarrays containing the new data. In this Section, we show that the EMA strategy can reduce the cost of array extensions significantly.

For the derivation of cost functions we compressed the multidimensional sparse array. Both CMA and EMA are assumed to be implemented as CDL, where $\langle h, o \rangle$ ($\langle h, o \rangle$) is the key for EMA and $\langle o \rangle$ is the key for CMA.

5.1 Parameters

Length of a node for EMA, $X_{EMA} = (kl + dl) \times kn$

Length of a node for CMA, $X_{CMA} = (ol + dl) \times kn$

where

kl = Length of key value of EMA

dl = Length of data value or fact data

ol = Length of key value of CMA

kn = Number of keys that can fit in a leaf node

All the lengths are in bytes. The cost functions are represented as the number of node access required because all the basic CPU operations can be executed in constant time.

Assumptions

To simplify the cost model we make a number of assumptions.

- (i) The length of dimensions extends in round robin manner for both CMA and EMA.
- (ii) The length of each dimension is equal and when extension occurs each of the dimensions are extended by equal length. We denote the length of dimension at i th extension as L_i .
- (iii) The records are uniformly distributed in the corresponding CMA or EMA. We denote the density of records by ρ both for CMA and EMA.

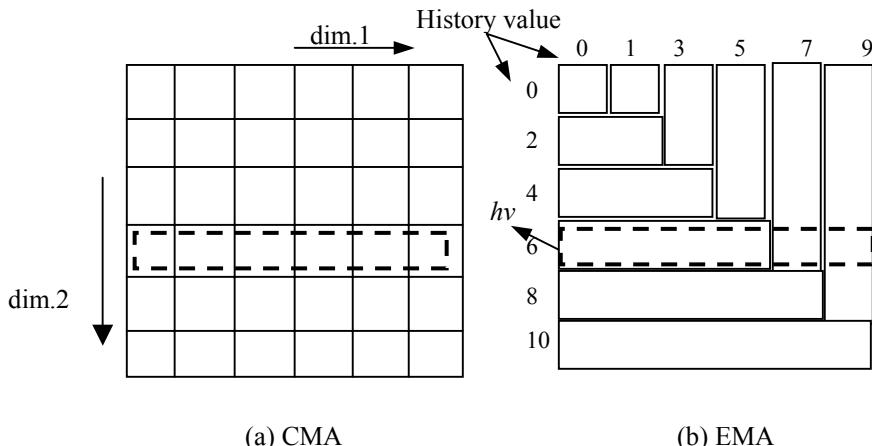


Fig. 4. The slice operation for CMA and EMA

5.2 Retrieval Cost

In CMA all offset values of the array elements are consecutive and linearized in a single data stream using the addressing function described in Section 2. Hence the range of candidate offset values for a query can be determined uniquely. But for EMA, the same data stream is distributed over different subarrays as shown in Fig. 4(b).

Cost function for EMA

For EMA, the records of all the subarrays having history value greater than hv (principal subarray) are checked. We assume that the principal subarray hv is found in the middle of the corresponding dimension. Hence the records to be checked at extension i in the *leaf nodes* of CDL will be $(L_i^n - (L_i/2)^n) \times \rho$ (due to the assumption 5.1 (i)). Hence the number of nodes accessed for retrieval from EMA is therefore

$$R^{EMA} = \lceil (1 - (1/2)^n) \times L_i^n \rho / kn \rceil \quad (1)$$

Cost function for CMA

In an n dimensional CMA, if the selection of the slice is along dimension n (i.e. subscript x_n is known) then all the candidate offsets are consecutive and the volume of the

range of the slice is L_i^{n-1} . This is explained with an example in the following. The addressing function of a 5 dimensional array is as follows

$$f(x_5, x_4, x_3, x_2, x_1) = s_1 s_2 s_3 s_4 x_5 + s_1 s_2 s_3 x_4 + s_1 s_2 x_3 + s_1 x_2 + x_1$$

Let us consider $s_j=L$ for $j=1,\dots,n$ (Assumption 5.1 (ii)). If $L=5$ and x_5 is known (say, $x_5=0$, and $x_j=0,\dots,L-1$ for $j=1,\dots,4$) then the candidate offset values in the slice are consecutive in the range 0 to 624 (total 625 offsets) out of 3125 offsets which is L^4 (i.e. 5^4). If x_1 is known (say, $x_1=0$, and $x_j=0,\dots,L-1$ for $j=2,\dots,5$) then the candidate offset values in the slice are in the range 0 to 3120 (total 3121 offsets) out of 3125 offsets. Hence the volume of the candidate offset values is determined by $L^5 - (L-1)$. If the subscript x_2 is known then the volume of the candidate range of offsets is $L^5 - L(L-1)$. In general, if the subscript x_k ($1 \leq k \leq n$) is known then the volume of the corresponding range of offsets to be searched in the slice is $L^n - L^{k-1}(L-1)$. Hence the number of records to be searched in the corresponding CDL of CMA at extension i is given by $\{L_i^n - L_i^{k-1}(L_i-1)\} \times \rho$ and number of nodes to be searched is given by $\lceil \{L_i^n - L_i^{k-1}(L_i-1)\} \times \rho / kn \rceil$ in the corresponding CDL of CMA.

The retrieval cost for CMA

$$\lceil L_i^n - L_i^{k-1}(L_i-1) \times \rho / kn \rceil \quad (2)$$

From equation (2) it can be realized that the retrieval cost for CMA is greatly dependent on the known dimension k .

5.3 Extension Cost

To extend the CMA the entire array has to be reorganized and all the offset values will be changed. For example, Fig. 5(a) shows the offset values of a 2 dimensional CMA. When the CMA is extended in dimension 1 (shown in Fig. 5(b)) the offset values are changed. Since the offset values are subject to change while reorganizing the CMA hence all the leaf nodes have to be faced to recalculate the offsets for CMA in the corresponding CDL.

If at the i th extension, Z_i is the number of leaf nodes needed for the extension then, the cost of extension E_{xt}

$$E_{xt}^{EMA} = Z_i \text{ and } E_{xt}^{CMA} = 2S_i + Z_i$$

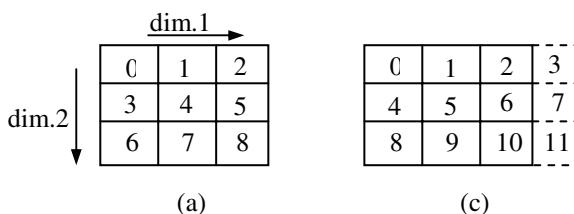
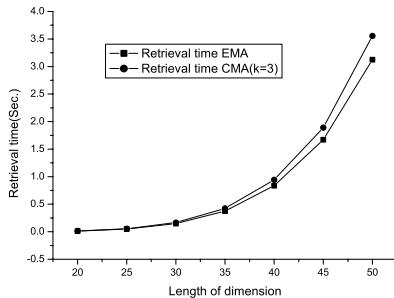


Fig. 5. Extension realization of a 2 dimensional CMA

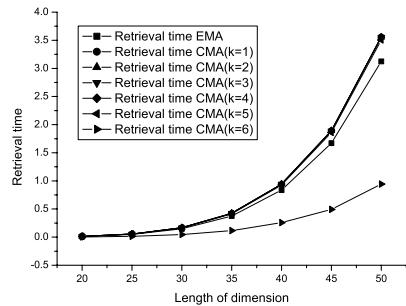
where

S_i = Number of leaf nodes before i th extension of the corresponding CDL; S_i is determined by $S_i = \lceil (L^n \times \rho) / kn \rceil$ (because of assumption 5.1 (ii)).

Z_i = The number nodes for i th extension; $Z_i = \lceil (L^{n-1} \times \rho) / kn \rceil$ (because of assumption 5.1 (i)(ii))



(a) Retrieval cost comparison for CMA and EMA($k=3$)



(b) Retrieval cost comparison for CMA and EMA ($k=1\dots 6$)

Fig. 6. Retrieval cost comparison for CMA and EMA

For extending CMA, it requires to reorganize the array and rewrite both existing and new data elements. The S_i leaf nodes need to be faced to recalculate the new offsets due to the extension. The factor of 2 accounts facing both the read and write operations of the existing nodes.

The size of the extended subarray for EMA can be calculated by knowing the size of each dimension except the dimension to be extended. The difference of extension cost between the two strategies is referred to as *Extension Gain* (EG)

$$EG = E_{xt}^{CMA} - E_{xt}^{EMA} = 2S_i \quad (3)$$

From equation (3) we can see that there is tremendous extension gain using EMA.

6 Experimental Results

We have constructed a prototype system having the parameter values shown in Table1 placing the *leaf nodes* of CDL in secondary storage and *head* on main memory. The test results for slice operation are analyzed in this Section. All the tests are run on a machine (SUN Enterprise 4500) of 1.05 GHz and 48 GB of memory having disk page size P=8KB.

Table 1. Parameter values for the prototype system

kl and ol	dl	kn	P	n	L	ρ
8	8	512	8192	6	20-50	0.00025

6.1 Retrieval Cost Comparison Between CMA and EMA

Fig. 6(a) shows the retrieval cost for known dimension $k=3$. The known dimension k has no effect for EMA but it has large effect in CMA. The retrieval cost for different values of k ($k=1,\dots,5$ and 6) is shown in Fig. 6(b). The retrieval cost for CMA is superior to EMA only for $k=6$ and for all the other values of k ($k=1,\dots,5$) EMA is superior to CMA for retrieval cost. This is because all the candidate offset values are consecutive for $k=6$ and the search volume is determined by L^5 and for other values of k ($k=1,\dots,5$) the candidate records are not consecutive and hence the search volume increases. It can be concluded that the retrieval performance for CMA is dependent on the known dimension k and it has better performance only if $k=6$ on the other hand EMA performance is independent of known dimension.

6.2 Extension Cost Comparison Between CMA and EMA

Fig. 7(a) shows the relative cost of extension for CMA and EMA for the system. The extension gain is shown in Fig. 7(b); the extension gain increases for increasing length of dimension. We put initial length of each dimension as 20 and then increased it accordingly up to 50. As can be seen from Fig. 7(a) that the extension cost for CMA is larger and it is noted from our experiment that the extension cost for CMA is on average 2.8 times more than that of EMA. The MOLAP extension can be achieved efficiently in this scheme. Fig. 7(b) shows the over all gain for each extension. The extension gain increases with the length of dimension. This is because the extension gain is $2S_i$ and if S_i increase the extension gain increases.

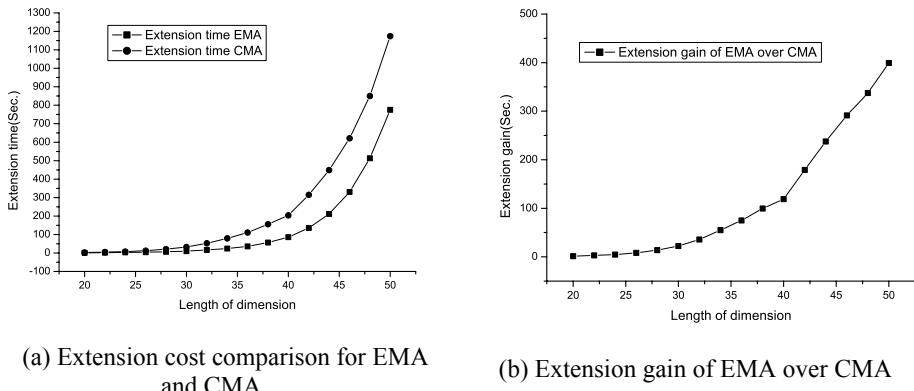


Fig. 7. Experimental result of extension cost comparison for EMA and CMA

7 Related Works

So far the authors know there is only one work [9] except our own work [8] on extendible multidimensional arrays for database applications. The extendible array is employed in [9] to extend the array and it only treats an organization scheme of the history tables. It does not concern with the actual data elements stored in the body of

the extendible array. All the elements occupy the physical storage, and the sparseness of the array elements is not considered. In MOLAP systems, the processing in the array based algorithm is done on a chunk by chunk basis [5][10]. A chunk is a unit of processing used in this algorithm and compressed on disk when more than a certain number of cells are empty. Again, to compute the data cube in chunk based algorithm it needs many chunks to be visited for multidimensional array [11] and computes multiple views simultaneously which consumes lot of main memory. We overcome this memory inefficiency in the array based algorithm by introducing a new method of encoding records of the array.

[12] attempts to solve the sparsity problem using the ROLAP approach, which employs a sort-based method developed in [13], where multiple aggregations are overlapped in a pipeline fashion after sorting tuples. The sparsity problem is handled in [14] by compressing the array using hashing method.

All the MOLAP [5][10][11][14] uses conventional multidimensional arrays as their basic data structure and the extension of the array is not handled. More over the sparsity of the array elements are not handled there. In our scheme the sparsity is handled effectively. Moreover it is shown that the array extension can be performed with very small cost comparing to the conventional multidimensional arrays.

8 Conclusion

We propose and evaluate an efficient MOLAP implementation to manage array extension by keeping track of the extension subarrays as opposed to the conventional multidimensional arrays. Our performance result shows that we can extend the MOLAP system employing extendible array and huge savings can be achieved for extension. The large reorganization cost that is incurred in conventional multidimensional array can be reduced and extendible MOLAP system can be constructed. We believe our scheme can also be applied to the multidimensional database implementations effectively specially for data warehousing applications for multidimensional analysis.

References

1. J. Gray, A. Bosworth, A. Layman, and H. Pirahesh, “Data cube: A relational aggregation operation generalizing group-by, cross-tabs and sub-totals”, Technical Report MSR-TR-95-22, Microsoft Research, Advance Technology Division, , Redmond, 1995.
2. V. Harinarayan, A. Rajaraman and J. D. Ullman, “Implementing data cube efficiently”, Proc. of ACM-SIGMOD Conf. Management of data, pp.205-216, 1996.
3. H. Gupta, V. Harinarayan, A. rajraman and J.D. Ullman, “Index selection for OLAP”, Proc. of ICDE, pp. 208-219, 1997.
4. Y. Kotidis and N. Roussopoulos, “An alterntive storage organization for ROLAP aggregation views based on cubetrees”, Proc. ACM-SIGMOD conf. Management of data, pp. 89-99, 1998.
5. Y. Zhao, P. M. Deshpande and J. F. Naughton, “An array-based algorithm for simultaneous multidimensional aggregates”, In Proceedings of the ACM SIGMOD Conference on Management of Data, pp.159-170, 1997.

6. J. Li and J. Srivastava, "Efficient aggregation algorithms for compressed data warehouses", IEEE Transaction on Knowledge and Data Engineering, 14(3), pp. 515-529, 2002.
7. Otoo E. J and T. H. Merrett, "A storage scheme for extendible arrays", Computing, Vol. 31, pp.1-9, 1983.
8. K. M. Azharul Hasan, M. Kuroda, N. Azuma, T. Tsuji, K. Higuchi, "An extendible array based implementation of relational tables for multidimensional databases", Proc. of Data warehousing and Knowledge Discovery (DaWak'05), pp.233-242, 2005.
9. D. Rotem and J. L. Zaho, "Extendible arrays for statistical databases and OLAP applications", Proc. of SSDBM'96, pp. 108-117, 1996.
10. S. Sarawagi and M. Stonebraker, "Efficient organization of large multidimensional arrays", Proc. of ICDE, pp. 328-336, 1994.
11. A. Shukla, P. M. Deshpande and J. F. Naughton, "Materialized view selection for multi-dimensional datasets", In Proc. of VLDB Conf., pp.488-499, 1998.
12. K. A. Ross and D. Srivastava, "Fast computation of sparse data cubes", In Proc. of VLDB conf., pp. 116-125, 1997.
13. S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, and S. Sarawagi, "On the computation of multidimensional aggregates", Proc. of VLDB, pp. 506-521, 1996.
14. S. Muto and M. Kitsuregawa, "Improving main memory for array-based data cube computation", Proc. of workshop on Data warehousing and OLAP, pp. 28-33, 1998.

Monitoring Heterogeneous Nearest Neighbors for Moving Objects Considering Location-Independent Attributes

Yu-Chi Su¹, Yi-Hung Wu², and Arbee L.P. Chen³

¹ Department of Computer Science, National Tsing Hua University, Taiwan, R.O.C.

² Department of Information and Computer Engineering, Chung Yuan Christian University, Taiwan, R.O.C.

³ Department of Computer Science, National Chengchi University, Taiwan, R.O.C.
alpchen@cs.nccu.edu.tw

Abstract. In some applications, data may possess both location-dependent and location-independent attributes. For example, in a job database, each job can be associated with both location-dependent attributes, e.g., the location of the work place, and location-independent ones, e.g., the salary. A person who uses this database to find a job may prefer not only a shorter distance between his/her house and the work place but also a higher salary. Therefore, a query with both concepts of “shorter distance” and “higher salary” should be considered to meet the user’s needs. We call it the heterogeneous k-nearest neighbor (HkNN) query in distinction from the traditional k-nearest neighbor (kNN) query in the spatial domain, which only concerns location-dependent attributes. To our knowledge, this paper is the first work proposing a generic framework for solving the HkNN query. We propose an efficient approach based on the bounding property for the HkNN query evaluation. Furthermore, we provide an update mechanism for continuously monitoring the HkNN queries in a dynamic environment. Experimental results verify that the proposed framework is both efficient and scalable.

1 Introduction

The continuous k-nearest neighbor query processing over moving objects, which aims at retrieving the k objects closest to a query point, has been studied for several years [2][3][4][5][7][8][9]. However, in many applications, a moving object may also have attributes that are irrelevant to its location. The following are two examples drawn from different applications. Example 1 illustrates a scenario over static objects, while Example 2 is for moving objects in high dimensions.

Example 1. A person wants to buy shoes of a special brand. He/she plans to visit the best k shoe stores in order of the shoes’ price and the expense for traveling to the store. Here we assume that the traveling expense for one distance unit (in kilometer) is 100 (in dollar) and the traveling expense equals the distance unit multiplied by the cost per distance unit. The total cost for buying new shoes at a store can be formulated

as: Total cost = Price (\$) + Traveling expense (\$). The person's need is to find the k stores with the minimal values as a function of the price and the distance from the person to the store. Motivated by this kind of applications with queries on both location-dependent and location-independent attributes, we propose the heterogeneous k -nearest neighbor (HkNN) query.

Example 2. Consider another example for HkNN queries over multi-dimensional data. In a digital library, each subscriber has a profile that records the degrees of his/her preference in various fields. Each profile can be considered as an object in a multidimensional space in which every dimension stands for a field. Also, each profile has several location-independent attributes, like income, age, and so on. If a service provider would like to publish a new electronic magazine focusing on young people, an HkNN query that retrieves the best k subscribers with smaller ages and profiles similar to the magazine content will be launched. The HkNN query here is to find the k subscribers with the minimal values as a function of the age and the distance from the profile to the magazine content.

Previous work on monitoring continuous k -nearest neighbor queries over moving objects can be broadly divided into two categories. In the first category, the motion patterns of objects are assumed to be known or predictable [2][5][7]. The second category does not assume the motion patterns of objects [3][4][8][9]. There are two reasons that the previous approaches to the k NN problem are not suitable for the HkNN query: (1) an intuitive way is to regard the location-independent attribute as an extra dimension in the multidimensional space constituted by the location-dependent attributes. After that, the total cost formula is used as the distance function in order to apply the previous approaches. Unfortunately, mixing two kinds of attributes with different domain sizes makes it difficult to build an index for efficient query processing. Moreover, if the operation “ $-$ ”, “ $*$ ”, or “ $/$ ” is adopted in the total cost formula, the distance function will not satisfy the triangle inequality and the pruning techniques will fail. More detailed discussions in this aspect can be found in [6]. (2) The previous approaches cannot deal with different operations with a single index. Therefore, a generic framework supporting the four operations and an efficient index structure are required.

In this paper, we propose a generic framework for efficiently processing the HkNN queries over moving objects and focus on the HkNN problem in which only one location-independent attribute is considered. The formulas in different applications to compute the total cost for each object can be generalized as $\text{total cost} = V_COST$ (from the location-independent attribute) $\text{op } D_COST$ (from the location-dependent attributes), where the operator op can be one of the four operations “ $+$ ”, “ $-$ ”, “ $*$ ”, or “ $/$ ”, as user needs. Moreover, we consider the situations like Example 2, in which both the location-dependent and location-independent attributes of objects may change with time. In the remainder of this paper, for ease of presentation, we illustrate our methods only for the HkNN queries with op “ $+$ ”. The considerations for the other ops (“ $-$ ”, “ $*$ ”, “ $/$ ”) are similar and can be found in [6].

Our method of HkNN query evaluation proceeds in two steps. In the first step, arbitrary k objects near the query are selected into the answer set. For each of the k objects, the total cost is computed and the one with the worst total cost is called the *target object*. Since these k objects may not be the correct answers, we then check if any other object has a total cost better than that of the target object. Based on a *bounding property*, the total cost of the target object is used to compute two

bounds - *value bound* and *distance bound*, which respectively indicate the upper bounds of the V_COST and D_COST for all the objects with better total costs. In other words, these bounds limit the search space of the HkNN query to a *safe region* for the remaining objects, in which all the objects with better total costs are located. In the second step, only the objects inside the safe region are retrieved and checked. Once an object with a better total cost is found, it is added into the answer set with the target object discarded. After that, a new target object comes out and the two bounds can be lowered to further reduce the search space. The second step is repeatedly executed until there is no object in the safe region. In this way, a large amount of unnecessary computation can be avoided. The proposed framework also supports continuous HkNN queries for a dynamic environment where the objects may continuously update their location-dependent or location-independent attribute values. In our approach, while receiving an object update, only the queries that can be affected are reevaluated and thus unnecessary computation on irrelevant queries can be avoided.

The rest of the paper is organized as follows. In Section 2, basic definitions and data structures are described. Section 3 depicts the techniques for HkNN query processing. Section 4 illustrates how to handle the updates with our index structure in a dynamic environment. Section 5 shows the experimental results and Section 6 concludes the paper with some future works depicted.

2 Basic Definitions and Data Structures

2.1 Basic Definitions

In this paper, we address the HkNN problem involving only one location-independent attribute. In the following, we first define several terms and the HkNN problem. Table 1 shows the symbols and functions used in this section.

Table 1. Notation and their definitions

Notation	Definitions
O	The set of moving objects
o	A moving object in O
q	The heterogeneous k -nearest neighbor query
$dist(o, q)$	The Euclidean distance between object o and query q
op	The user-defined operator, can be $+$, $-$, $*$, or $/$

Definition 1 (Object and query representations). A moving object o is represented as $o(v, p)$, where v denotes the value of its location-independent attribute and $p = \langle c_1, c_2, \dots, c_n \rangle$ denotes its coordinates, i.e., the location-dependent attribute of o (n is the number of dimensions). An HkNN query q is represented as $q(op, p')$, where op denotes one of four operators and $p' = \langle c'_1, c'_2, \dots, c'_n \rangle$ denotes its coordinates, i.e., the location-dependent attribute of q .

Definition 2 (D_COST). Let *d-factor* be the cost for one distance unit. The cost of the location-independent attribute for an object $o(v, p)$ with respect to query $q(op, p')$ is defined as: $D_COST = dist(o, q) * d\text{-factor}$.

Definition 3 (V_COST). The cost of the location-independent attribute for object o (v, p) is defined as: $V_COST = v$.

Definition 4 (T_COST). Given object $o(v, p)$ and HkNN query $q(op, p')$, we refer to the total cost of o with respect to q as: $T_COST(o, q) = V_COST \cdot op \cdot D_COST$.

Most applications in the real world demand the objects that are closest to a given query. Therefore, our framework is designed to prefer the objects with smaller values of D_COST no matter which operator is adopted. Applications with **op** “+” and “*” also have the nature to prefer objects with smaller values of V_COST and their goal is to find the k objects with the smallest values of T_COST. In contrast, applications with **op** “-” and “/” require objects with larger values of V_COST but smaller values of D_COST and their goal is to obtain the k objects with the largest values of T_COST. As a result, the definition of the HkNN problem is given as below by considering the two classes of operators.

Definition 5 (HkNN). Given a set of moving objects O , (1) if op of query q is “+” or “*”, the **HkNN** answers of q are defined as: $HkNN(q) = \{o \in O \mid T_COST(o, q) \leq T_COST(o_k, q)\}$, where o_k is the object with the k -th smallest value of T_COST in O . (2) If op of q is “-” or “/”, the HkNN answers of q are: $HkNN(q) = \{o \in O \mid T_COST(o, q) \geq T_COST(o_k, q)\}$, where o_k is the object with the k -th largest value of T_COST in O .

In this paper, the d-factor is set as 1 for simplicity and, from now on, we illustrate our approach using the case that the operator of the launched query is “+”. The proposed framework is developed for a general environment, i.e., the motion patterns of objects and queries are unpredictable. The following illustration uses 2D data, but the proposed approach can be applied to the environment with arbitrary dimensionality.

2.2 Data Structures

Before introducing our approach, we first present four data structures that will be used.

Object Table. Each object o is associated with a set of attributes, including the object ID id_o , the location-dependent attribute p_o , the location-independent attribute v_o , and a set S_o of queries the answer sets of which contain o .

Hierarchical Aggregate Grid Index. Previous work [9] uses the hierarchical grid structure to reduce the performance degradation caused by skewed data. Differently, we here adopt it to speed up the HkNN query processing by enclosing location-independent attribute information in each cell of the hierarchical grid structure. Note that we can also embed this kind of information in each internal node of an R-tree structure in a similar way. In this paper, we adopt the grid-based structure because of its efficient construction and maintenance in a dynamic environment.

The hierarchical aggregate grid index has several levels of grids and consists of two types of cells: *basis cells* and *index cells*. Index cells form a hierarchy, where each index cell points to smaller cells it covers at the lower level (called the *sub-cells*). The bottom level of the hierarchical grid structure is composed of basis cells, the smallest unit in the index. Let $C_{i,j}$ denote a cell, which can be an index cell or a basis cell, at column i and row j of grid level C . Moreover, each basis cell $X_{i,j}$

(Assume the bottom level is level X) with equal length δ is associated with one bucket that stores every object ID with coordinate (x, y) , where x is in the range $[i\delta, (i+1)\delta]$ and y is in the range $[j\delta, (j+1)\delta]$. Every object or query moving from (x_{old}, y_{old}) to (x_{new}, y_{new}) is deleted from the bucket of cell $(\lfloor x_{old} / \delta \rfloor, \lfloor y_{old} / \delta \rfloor)$ and inserted into the bucket of cell $(\lfloor x_{new} / \delta \rfloor, \lfloor y_{new} / \delta \rfloor)$. In addition, to enclose location-independent attributes of objects in each cell, both the basis cell and the index cell are associated with three pieces of aggregate information: *min*, *max*, and *count*. For a basis cell, the *min* (*max*) indicates the minimal (maximal) value of location-independent attributes for the objects in the cell and similarly the *count* represents the total number of objects enclosed in the cell. For an index cell, *min* (*max*) is the minimum (maximum) of all the *min* values attached on its sub-cells, while the *count* keeps the sum of all the *count* values attached on the sub-cells. Figure 1 shows an object table and a 3-D illustration of the hierarchical aggregate grid index. Both types of cells are associated with the aggregate information (a, b, c) , where a , b , and c represent *min*, *max* and *count*, respectively. Moreover, every basis cell is associated with an object bucket storing the IDs of the enclosed objects together with the links, pointing to the corresponding entries in the object table.

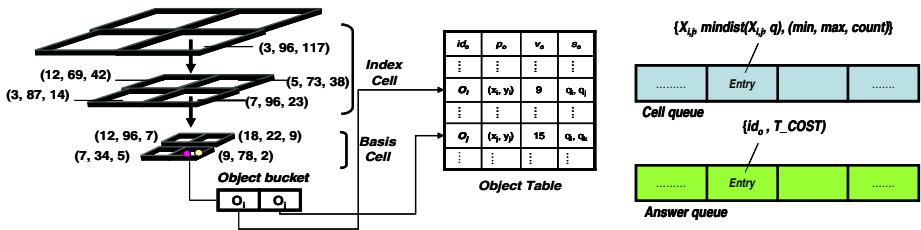


Fig. 1. Hierarchical aggregate grid index and object table

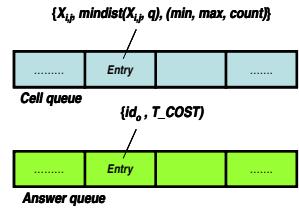


Fig. 2. Cell queue and answer queue

Cell Queue. Let $mindist(X_{i,j}, q)$ be the minimal distance between cell $X_{i,j}$ and query q , i.e., the minimal possible distance between any objects in cell $X_{i,j}$ and q . Whenever a query q is evaluated (or reevaluated), a cell queue CQ is created. Each entry in CQ stores a cell $X_{i,j}$ (index cell or basis cell), $mindist(X_{i,j}, q)$, and the aggregate information: *min*, *max*, *count* of $X_{i,j}$. The entries in CQ are kept in ascending order of $mindist(X_{i,j}, q)$.

Answer Queue. Each query q is associated with an answer queue AQ to maintain the current set of query answers. Each entry in the answer queue keeps an object ID and its T_COST with respect to q . The entries in AQ are kept in ascending order of T_COST . Figure 2 shows both kinds of queues for query q .

3 Efficient Evaluation of H k NN Queries

3.1 Problem Characteristic and Approach Overview

A naive method for processing the H k NN query is to compute the total cost of each object and then select the k objects with the smallest values as the answers. Nevertheless,

it is inefficient if the number of objects is very large. Consider an alternative method using the data structures in Section 2.2. To evaluate query q , all the cells in the hierarchical aggregate grid structure can be pushed into the cell queue CQ in ascending order of their minimal distances from q . From CQ , the cells are explored one by one to obtain the first k objects. The k objects are then put into the answer queue AQ in ascending order of their total costs. We call the object with the largest total cost as the *target object* and its total cost as the *target cost*. Given AQ and the target cost, we apply the following property to prune the remaining objects in CQ , which cannot be the HkNN answers.

Property 1 (Bounding property). *For an object o in cell $X_{i,j}$ of CQ , if its total cost is not larger than the target cost, then its D_COST is not larger than the target cost and its V_COST is not larger than the target cost minus $\text{mindist}(X_{i,j}, q)$.*

Proof. Let r be the target object. Moreover, let D_z , V_z and T_z respectively denote the D_COST , V_COST and T_COST of object z . Since $V_o \geq 0$ and $T_o = D_o + V_o \leq T_r = D_r + V_r$, we have that $D_o \leq D_r + V_r - V_o \leq D_r + V_r - 0 = T_r$. Similarly, because $D_o \geq \text{mindist}(X_{i,j}, q)$, we have that $V_o \leq D_r + V_r - D_o \leq D_r + V_r - \text{mindist}(X_{i,j}, q) = T_r - \text{mindist}(X_{i,j}, q)$.

From this property, two upper bounds, *value bound* and *distance bound*, are obtained and used to prune the cells in CQ , in which all the objects violate either of them. The objects in a cell are retrieved to compute their total costs if and only if the cell satisfies both bounds. AQ and the target object are then updated. Since the new target object results in a smaller target cost, the bounds can be tighter and tighter. Eventually, CQ will become empty and AQ will have the k objects with the smallest total costs. The pruning mechanism using the two bounds will be described in Section 3.3.

3.2 Step 1: Retrieving the First k Objects

Given an HkNN query q , we discuss our first step of query evaluation for q in this section. Let CQ and AQ respectively denote the cell queue and the answer queue of q . To begin with our approach, the cells at the highest level of the index are first visited and then inserted into CQ in ascending order of their minimal distances from q . Next, our approach starts to retrieve the first entry in CQ . If it is an index cell, its sub-cells

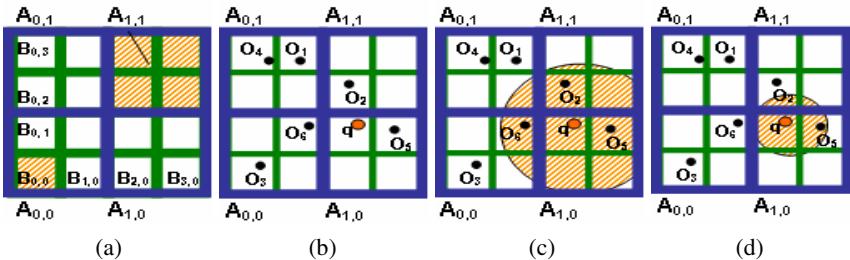


Fig. 3. An example of query evaluation using two-level hierarchical aggregate grid index

are inserted into CQ according to their minimal distances to q . This process is repeated until the first entry of CQ is a basis cell. In this case, the total costs of all the objects in this cell are computed. Their object IDs and total costs are then inserted into AQ in

ascending order of their total costs. This step terminates while the number of objects in AQ is not less than k . Note that if the number of objects in AQ is more than k , only the k objects with the smallest total costs are chosen and used in the next step.

Example 3. Figure 3 shows an example of query evaluation using a two-level hierarchical aggregate grid index. Let A and B represent the higher level and the lower level of grids, respectively. The bigger shaded cell ($A_{1,1}$) and the smaller one ($B_{0,0}$) are shown in Figure 3 (a). In Figure 3 (b), given an H2NN query q , the search for the first k objects is executed (let k be 2). Initially, the cells of the higher level are added into CQ in ascending order of their minimal distances from q , i.e., $CQ = \{A_{1,0}, A_{1,1}, A_{0,0}, A_{0,1}\}$. Then, the first entry $A_{1,0}$ is retrieved and its sub-cells $B_{2,0}, B_{2,1}, B_{3,0}, B_{3,1}$ are inserted into CQ . Since all the counts in $B_{2,0}, B_{2,1}$, and $B_{3,0}$ are zero, these cells are ignored to avoid unnecessary computation. After that, CQ becomes $\{A_{1,1}, B_{3,1}, A_{0,0}, A_{0,1}\}$. Next, $A_{1,1}$ is retrieved in the same way to update CQ as $\{B_{2,2}, B_{3,1}, A_{0,0}, A_{0,1}\}$. $B_{2,2}$ is then retrieved and the total cost of o_2 is computed (let its total cost be 1.6). As a result, o_2 is inserted into AQ with its total cost, i.e., $AQ = \{(o_2, 1.6)\}$. This step terminates after the next entry $B_{3,1}$ updates AQ as $\{(o_5, 0.7), (o_2, 1.6)\}$. At this time, CQ becomes $\{A_{0,0}, A_{0,1}\}$.

3.3 Step 2: H k NN Search with Pruning Mechanism

The second step for query evaluation is to iteratively find the objects with smaller total costs and replace the target object to further reduce the search space until the safe region is empty. A naïve method is to sequentially examine all the objects located in the circle centered at q with radius D (for short, from now on, sometimes we denote the distance bound and value bound as D and V , respectively). Obviously, this method can be inefficient if more basis cells than necessary are checked. Therefore, we design a pruning mechanism to skip the objects that are unable to be the final answers. The pruning mechanism keeps retrieving the first entry of CQ to check whether any object in it satisfies the two bounds. The following are the two pruning techniques we adopt.

Pruning by Distance Bound. Cell $X_{i,j}$ in CQ needs to be checked if and only if it overlaps with the circle centered by q with radius D , i.e., $mindist(X_{i,j}, q) \leq D$. In this case, cell $X_{i,j}$ may contain an object whose distance from q is less than D and therefore should be further checked by the value bound. Otherwise, $X_{i,j}$ and all the remaining cells in CQ can be discarded because all the objects inside them cannot be the final answers (their distances from q must be larger than D).

Pruning by Value Bound. For each cell $X_{i,j}$ in CQ , the value bound is computed as $V = c_t - mindist(X_{i,j}, q)$, where c_t denotes the target cost. If the min value associated with $X_{i,j}$ is larger than V , $X_{i,j}$ can be omitted. Otherwise, there are two cases to consider. If $X_{i,j}$ is an index cell, its sub-cells are retrieved from the hierarchical aggregate grid structure and inserted into CQ . On the other hand, if $X_{i,j}$ is a basis cell, the total costs of all objects in it are computed and then the objects with total costs smaller than c_t are inserted into AQ . Finally, only the k objects with the smallest total costs are left in AQ and then a new target object can be found. In this way, new and smaller distance bound and value bound will be obtained.

Example 4. Figure 3(c) and 3(d) show an example for the second step of query evaluation. We assume that $\text{mindist}(A_{0,0}, q)$, $\text{mindist}(B_{1,1}, q)$, and $\text{dist}(o_6, q)$ are 0.8, 0.8, and 0.9, respectively. In addition, we assume that the min value of $A_{0,0}$ and $B_{1,1}$ are 0.6 and 0.4, respectively and the V_COST of o_6 is 0.1. In Figure 3(c), the radius of the shaded circle centered by q is equal to D . Following Example 3, AQ is $\{(o_5, 0.7), (o_2, 1.6)\}$ and $A_{0,0}$ is first examined. The value bound for $A_{0,0}$, i.e., $1.6 - \text{mindist}(A_{0,0}, q)$, is larger than the min value of $A_{0,0}$, its sub-cells are inserted to CQ . Again, since $B_{0,1}$ and $B_{1,0}$ are empty, CQ becomes $\{B_{1,1}, A_{0,1}, B_{0,0}\}$. The value bound of $B_{1,1}$ is computed in the same way and equals to 0.8, which is larger than the min value of $B_{1,1}$. Therefore, the object o_6 in $B_{1,1}$ is retrieved to replace o_2 as the new target object (o_6 has a total cost smaller than that of o_2), i.e., $AQ = \{(o_5, 0.7), (o_6, 1)\}$. Then, the shaded circle shrinks because D decreases. We show the new circle in Figure 3(d). Next, we get the next entry $A_{0,1}$. From Figure 3 (d), we can observe that it does not overlap with the shaded circle. In other words, the minimal possible distance of any objects in $A_{0,1}$ is larger than D and therefore the remaining cells in CQ can be skipped. The pruning process ends and the final answers of q are $\{o_5, o_6\}$.

4 Continuous Update of H_kNN Query Answers

Another contribution of our framework is the update mechanism for maintaining the answers of queries in a dynamic environment, where objects and queries may update their status continuously. For objects, both of their positions and location-independent attribute values can be updated, while for queries, only their positions may be updated. In the following, we illustrate our approach for continuously processing a single object update.

To maintain the H_kNN query answers while receiving an object update, a naive way is to reevaluate all the H_kNN queries. However, it is impractical if there are a large number of queries registered. Obviously, a better solution exists if we can find out all the queries that will be influenced by the updated object. Let cost_k denote the largest total cost of the object in the answer queue of q after the query evaluation. Queries whose answers are affected by the updated object can be divided into two classes. In the following, we discuss them respectively.

Case 1. The first class consists of the queries whose answer sets do not include the updated object. For a query q belonging to this class, its cost_k will not be changed immediately in response to the updated object. Based on the bounding property, an object o_i can be used to replace the target object of q only if both the D_COST and V_COST of o_i satisfy the distance bound and value bound of q , respectively. Motivated by this, for each query, we refer to the *distance influence region* as a region where all the objects satisfy the distance bound of q . Similarly, we define the *value influence region* to identify all the objects satisfying the value bound of q . The total cost of o_i is compared with cost_k only if the two regions of query q both contain o_i . If object o_i has a smaller total cost, it is a new answer of q and cost_k is updated. Otherwise, the answer queue of q remains unchanged. More details of the two regions are discussed in the following:

Distance Influence Region. The region composed of all the basis cells that intersect the circle centered at q with radius D is called the *distance influence region*. Clearly, a basis cell may be covered by multiple distance influence regions of different queries. To identify these queries, each basis cell of the grid is associated with a query bucket containing the queries whose distance influence regions intersect it. Figure 4 shows this data structure.

Value Influence Region. We adopt the B+-tree, named *the value influence region tree*, to organize the value influence regions of all the queries. Figure 5 shows an example. In this tree, the value bounds of queries are the access keys attached on the nodes. Moreover, each internal node (or called *index node*) is constructed after the split or merging of tree nodes. On the other hand, each leaf node (or called *data node*) keeps a set of queries. Note that each leaf node has a pointer, pointing to the sibling next to it. Assume that there is an object o_i with $V_COST = 5$. To find the queries whose value influence regions contain object o_i , a range search for all the queries with $V \geq 5$ is launched on the value influence region tree. Since the amount of returned queries may be huge, we first check the query bucket of the cell containing the updated object to find the queries whose distance influence regions contain o_i . Then, these queries are checked one by one to see whether their value influence regions contain o_i by launching exact searches on the value influence region tree.

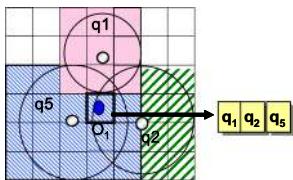


Fig. 4. Distance influence region

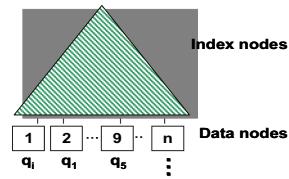


Fig. 5. Value influence region tree

Case 2. The second class includes the queries that answer queues of which contain the updated object. In this case, $cost_k$ may be changed even when the answer queue has the same set of objects. According to the possible changes of $cost_k$, there are three situations to consider: The first situation is that $cost_k$ remains unchanged. This may occur due to two causes: (1) the updated object is not the target object and its new total cost is still not larger than $cost_k$ or (2) the updated object is the target object but its new total cost is unchanged although both the V_COST and D_COST are changed. For both causes, we only need to update the information about the updated object. The second situation is that $cost_k$ becomes smaller. This also occurs due to two causes: (1) the updated object is the target object and its total cost decreases. or (2) the updated object is the target object and its new total cost is smaller than the object with $k-1$ th smallest total cost in the answer set of q . Since both $cost_k$ and the order of answers may be changed, we need to update the two bounds and the two influence regions by using the new target object and $cost_k$.

The third situation is that $cost_k$ becomes larger. It happens in two conditions: (1) the updated object is the target object and the changed V_COST or D_COST makes its total cost larger or (2) the updated object is not the target object but belongs to the answer set of q , and its updated V_COST or D_COST let its total cost larger than

$cost_k$. In both conditions, since $cost_k$ becomes larger, the object with the $k+1$ th smallest cost may replace the updated object as a new answer of q . Therefore, we assume the updated object, denoted as o_t , to be the target object and invoke the HkNN re-evaluation to find the object with smaller total cost than o_t to replace it. Due to space limitation, more details about the HkNN reevaluation are discussed in [6].

Example 5. We illustrate an example for processing an object update using Figure 4 and Figure 5. Suppose the server receives an update record $\{o_1, x, y, v, x', y', v'\}$ representing o_1 moves from (x, y) to (x', y') and its value changes from v to v' . Assume $v' = 3$ and the updated $dist(o_1, q) = 15$. First, o_1 is deleted from the object bucket of the basis cell $X_{i,j}$ and then inserted into the basis cell $X_{i',j'}$, where i, j, i' , and j' are $\lfloor x/\delta \rfloor, \lfloor y/\delta \rfloor, \lfloor x'/\delta \rfloor$, and $\lfloor y'/\delta \rfloor$, respectively. Next, the new total cost of o_1 is computed. After that, the queries whose answer sets contains o_1 are found by checking the object table. Assume q_2 is found and the total cost of the target object of q_2 is 13. Since the new total cost of o_1 is 18, the $cost_k$ of q_2 becomes larger (from 13 to 18, the third situation of Case 2). Therefore, HkNN query reevaluation is invoked to update the answers of q_2 . Next, the algorithm finds the queries whose distance and value influence region both contain o_1 but their answer sets do not include o_1 (Case 1). As shown in Figure 4, the cell enclosing o_1 is covered in the distance influence regions of q_1, q_2 , and q_5 . Since query q_2 has been processed, it can be ignored here. Next, we check if the value influence region of q_2 or q_5 contains o_1 . A range query q_r , which searches the queries with $V \geq 3$ on the value influence region tree is issued. In Figure 5, because q_1 is linked to the leaf node with key = 2, i.e. the value bound of q_1 is 2, q_1 will not be in the returned results of q_r . Thus, o_1 is not in the value influence region of q_1 . On the contrary, q_5 with $V = 9$ is in the results of q_r . Since o_1 is contained in both the distance influence region and value influence region of q_5 , o_1 is probably qualified to be a new answer of q_5 . Therefore, the updated total cost of o_1 ($cost_1$, for short) needs to be compared with that of the target object of q_5 ($cost_e$, for short). If $cost_1 < cost_e$, the target object of q_5 is deleted from the answer set of q_5 and o_1 becomes a new answer. Then, the bounds and influence regions for q_5 are updated. Otherwise, q_5 is not affected by the update of o_1 .

5 Experiments

We compare the proposed method using the hierarchical aggregate grid index with a method using a one-level grid and a brute force method. For simplicity, we call the above three methods *HAG*, *OLG*, and *BF*, respectively. Similar to HAG, OLG is a method based on the bounding property proposed in this paper. The distinction of OLG from HAG is that it employs one-level grid structure instead of hierarchical grid index. One-level grid structure consists of equal-sized cells and each cell is associated with three pieces of aggregate information-min, max and count. In the first phase of evaluating an HkNN query q , OLG adopts the method proposed in [9] to find the kNN of q as the first k objects. Next, it pushes the cells overlapping with the circle centered by q with radius D into the queue of q and runs the pruning algorithm in the same way

as our approach. The brute-force method for comparison computes the total costs for all objects in the database and then choose the k objects with smallest total costs as the answers.

In all the experiments, we utilize a program modified from the *Network-based Generator of Moving Objects* [1] to generate a set of moving objects and queries. The generator outputs a set of objects with their values at every timestamp. The default mobility of objects and queries are 10% and 5%, respectively. Table 2 lists the parameters of the data sets, where the default values are bold and italic. All our experiments are performed for 100 time-stamps and the CPU time (in seconds) is reported after a work is completed. For HAG and OLG, the locations of objects (or queries) and the aggregate information of all cells are updated after receiving all the update records at each timestamp.

Table 2. Parameters of the data sets

Parameter	Range
Number of Object (K)	10, 50, 100 , 150, 200
Number of Queries (K)	1, 2, 4, 6 , 8, 10

Table 3. Grid structures with various cell sizes

OLG	HAG
32×32	4×4, 16×16, 32×32
64×64	4×4, 16×16, 64×64
128×128	4×4, 32×32, 128×128
256×256	4×4, 32×32, 256×256
512×512	4×4, 64×64, 512×512
1024×1024	4×4, 64×64, 1024×1024

5.1 Experimental Results

The first experiment evaluates how the performance of OLG and HAG will be affected by different grid size. Moreover, for HAG, the number of levels in the hierarchical grid index also has a great impact on its performance. To fairly compare the performance of the two methods with respect to the grid size, we fix the number of levels to 3 for HAG and vary the grid size of both methods from 32×32 to 1024×1024 . For OLG, this parameter indicates the grid size of the one-level grid index, while for HAG it represents the grid size for the grid at basis level in the hierarchical aggregate grid index (denoted as $2^b \times 2^b$). In addition, we fix the grid size of the top level to 4×4 (i.e. $2^2 \times 2^2$). The granularity of any index cell at a lower level is set to be $2^l \times 2^l$, where $l = \lceil (\log_2 b + 2)/2 \rceil$. Table 3 summarizes the granularities used in our experiments.

The experimental results are shown in Figure 6. Both methods achieve the best performance when the 128×128 grid is used, while the other grid sizes cause the two methods higher CPU costs. This is because their grid indices with fine granularities incur frequent updates, whereas coarse granularities result in linear searches on huge object buckets during accessing a cell. We also observe that 3-level HAG has better performance than OLG in all cases. The reasons are as follows: (1) After the first step of evaluation of query q , OLG puts all the cells intersecting with the circle centered at q with radius D into the cell queue in ascending order of their minimal distances from q . The computations of these minimal distances for all cells covered by the circle result in expensive costs. (2) Despite HAG have the same number of cells at basis level as that of the index in OLG, the good pruning effect of HAG helps pruning many index cells that do not cover any answer. Thus, unnecessary computations for

minimal distances for all basis cells can be avoided. (3) Moreover, although HAG requires more updates for maintaining aggregate information due to multiple-level grids, such updates (in both methods) can be done by batch processing at each timestamp. Therefore, the cost of updating hierarchical aggregate information can be limited. Based on the above observations, the remaining experiments are made using the 128×128 grid for HAG and OLG.

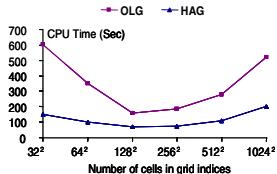


Fig. 6. Performance vs. Granularity

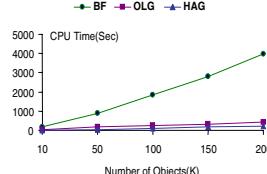


Fig. 7. Performance vs. N_O

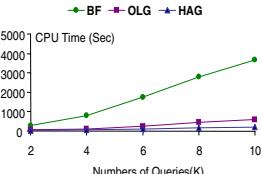


Fig. 8. Performance vs. N_Q

In the second experiment, we compare the performance of HAG, OLG and BF by varying the number of objects (denoted as N_O) from 10K to 200K. Figure 7 depicts that the CPU costs of all methods increase when N_O increases. Moreover, BF is much less efficient than the other two methods. The reason is that for each query, BF computes the total costs, in which expensive distance computations are involved, for all objects in the database. Furthermore, all the queries registered in the system are re-evaluated to keep the answers correct when the update requests are received. On the contrary, in all cases the execution times of OLG and HAG are relatively small. This is because both methods are designed based on the bounding property proposed in this paper, which helps greatly reducing the search space. Furthermore, they handle updates only on the queries whose answers may be affected by the updated objects instead of reevaluating all the registered queries. In Figure 7, we also observe that the performance of HAG is only a little better than OLG. The reason is that a larger N_O leads to the larger probability of a query reevaluation after the given object update. (The third situation in Case 2 for handling object updates) Similar result is shown in Figure 8, where the number of queries (denoted as N_Q) is varied. HAG has better performance than OLG. As mentioned in the first experiment, HAG possesses several advantages over OLG in processing a single query. Therefore, OLG is also more sensitive than HAG when N_Q increases. To sum up, in both figures, HAG is more efficient than OLG and BF. These experimental results verify that HAG achieves good scalability and efficiency.

6 Conclusion and Future Works

In this paper, we introduce the *heterogeneous k-nearest neighbor (HkNN)* query, a new paradigm considering both location-dependent and location-independent attributes over moving objects. HkNN queries are of nature interesting in many applications. To the best of our knowledge, this is the first work addressing the HkNN problem and providing an efficient approach for HkNN query evaluation. Based on the bounding property, our approach employs the *hierarchical aggregate index*,

which recursively aggregates the values of location-independent attribute in a hierarchy of cells, to quickly reduce the search space of the HkNN query. Furthermore, we developed an efficient update mechanism for continuously monitoring the affected HkNN queries during an object update and for maintaining the correctness of query answers. In particular, our approach can handle different types of operators with a single index. Our experimental results demonstrate the efficiency and scalability of the proposed techniques.

In our work, we process every query using a cell queue that employs the minimal distance between the query and each enclosed cell as its key. In other words, we adopt a distance-based technique to solve the HkNN problem in this paper. In the future, we plan to research from the aspect of location-independent attributes and then develop a hybrid mechanism that can adaptively determine from which aspect the query processing should start with.

References

1. T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, 6(2): 153-180, 2002.
2. G. S. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. *VLDB*, 2003.
3. M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. *SIGMOD*, 2004.
4. K. Mouratidis, D. Papadias, and M. Hadjieleftheriou, “Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring.” *SIGMOD*, 2005.
5. K. Raptopoulou, A. Papadopoulos, and Y. Manolopoulos. Fast nearest-neighbor query processing in moving –object database. *GeoInformatica*, 7(2):113-137, 2003.
6. Y. C. Su. Technique Report: Monitoring Heterogeneous Nearest Neighbors for Moving Objects Considering Location-Independent Attributes. <http://make.cs.nthu.edu.tw/people/Steffi/Technique.htm>, 2006.
7. Y. Tao, D. Papadias. Time-parameterized queries in spatio-temporal databases. *SIGMOD* Conference, 2002.
8. X. Xiong, M. F. Mokbel, and W. G. Aref. SEA-CNN: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. *ICDE*, 2005.
9. X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. *ICDE*, 2005.

Similarity Joins of Text with Incomplete Information Formats

Shaoxu Song and Lei Chen

Department of Computer Science
Hong Kong University of Science and Technology
`{sshaoxu,leichen}@cs.ust.hk`

Abstract. Similarity join over text is important in text retrieval and query. Due to the incomplete formats of information representation, such as abbreviation and short word, similarity joins should address an asymmetric feature that these incomplete formats may contain only partial information of their original representation. Current approaches, including *cosine similarity with q-grams*, can hardly deal with the asymmetric feature of similarity between words and their incomplete formats. In order to find this type of incomplete format information with asymmetric features, we develop a new similarity join algorithm, namely **IJoin**. A novel matching scheme is proposed to identify the overlap between two entities with incomplete formats. Other than *q-grams*, we reconnect the sequence of words in a string to reserve the abbreviated information. Based on the asymmetric features of similar entities with incomplete formats, we adopt a new similarity function. Furthermore, an efficient algorithm is implemented by using the join operation in SQL, which reduces pairs of tuples in similarity comparison. The experimental evaluation demonstrates the effectiveness and the efficiency of our approach.

1 Introduction

Similarity Join is an important operation in data cleaning and data integration [4]. It has been studied by various aspects and referred by a variety of names, including record linkage [7], entity identification [9] and approximate join [5]. The key issue is to identify whether two entities (e.g., relational tuples) are approximately the same[7]. Owing to the poor data quality with various errors caused by human factors and technique problems(e.g., database system problems), it is difficult to identify the same entities exactly by traditional join operation in SQL. For example, “International” and “Intenational” with spelling mistakes do not match exactly. By using *edit distance* [10], we can deal with the spelling mistake. Furthermore, due to various kinds of formats in representing information, it becomes even harder to detect approximate entities, such as different orders (“Shaoxu song” with given name first and “SONG, Shaoxu” with surname first). *Cosine similarity with q-grams* [6] is used in dealing with block orders, which is also effective in spelling mistakes. In addition to effectiveness, efficiency is another key issue. A recent work [2] applied a join operation in

SQL to identify entities with overlaps first, which reduced the times of comparisons between unrelated entities greatly. They proposed an efficient similarity join operators that can be used in many similarity functions.

However, the current existing approaches, including cosine similarity with *q-grams*, can hardly deal with the similarity between words and their incomplete formats, such as abbreviation, short word and incomplete information. For instance, there are various representing formats in bibliography references, including abbreviation (“VLDB” for “Very Large Databases”), short word (“Conf.” for “Conference”), incomplete information (“In VLDB” for “In Proceedings of VLDB”). Those incomplete formats have a common asymmetric feature that incomplete formats contain only partial information of their original representation, which make it difficult to find and quantify the similarity between words and their incomplete formats. Current *q-grams* approach cannot identify the abbreviation information, while *cosine similarity* is not effective in dealing with such asymmetric features. There is a solution by Rohit [11] which tries to handle the abbreviation case. However, it needs several attributes with hierarchies, such as, County, State, City and Street. So it is NOT a common solution without humans domain knowledge.

In this paper, we propose a novel text similarity join approach, **IJoin**, to address both the effectiveness and efficiency issues of identifying the similarity between text entities with incomplete formats in similarity joins. In text matching, we connect first letter of each word to reserve potential abbreviation information and enhance the importance of the first few grams to find high similarity between words and their short formats. We also consider the asymmetric features of similarity between entities with incomplete formats in similarity function. Our contributions in this paper are summarized as follows:

- (1) We propose a novel matching scheme to identify overlaps between text entities and their incomplete formats;
- (2) We design a similarity function which can calculate the similarity of entities with asymmetric features when incomplete formats exist;
- (3) We present an efficient implementation of our similarity joins algorithm which uses the join operation in SQL.

The rest of the paper is organized as follows. Section 2 summarizes text matching schemes, and introduces our IJoin matching approach. Section 3 presents our IJoin similarity function. In Section 4, we illustrate the basic and extended implementations of our IJoin approach. Section 5 reports the experimental results on the effectiveness and scalability of IJoin. Finally, we conclude in Section 6.

2 Distance Based Matching

In this section, we first summarize text matching schemes in similarity joins. Then, we illustrate our IJoin matching scheme, considering asymmetric features of incomplete information formats.

In similarity joins, we consider two relations (e.g. R and S) with common attributes (e.g. $R.A$ and $S.A$). The main issue is to find tuples in R and S with

exactly same or approximate values of the common attribute A . If the similarity between two tuples r and s from R and S respectively, satisfies the user specified similarity threshold η , then these two tuples will be the join result. Different from *edit distance* [10] based measure, we first map text strings in tuples to a set of elements as entity features. Based on the overlap of the sets, we calculate the similarity value by certain similarity functions (discussed in Section 3).

2.1 Existing Matching Approaches

One common text matching approach used in text retrieval, is to map a string to a set of word tokens [3]. For example, the string “Computer Science Department” can be mapped to a set of words, {‘Computer’, ‘Science’, ‘Department’}. The word token based matching scheme can identify the similarity of same entities with different representing orders, where each word is treated as a block. A block move in the string affects the mapping set slightly. The strings “Computer Science Department” and “Department of Computer Science” have high overlap in their mapping sets. However, the word based approach is not effective in dealing with spelling errors. A spelling error, such as “Computer”, may affect the similarity of strings significantly.

Another widely used matching method is *q-grams*, which cuts a string into several substrings of length q . For the same example, “Computer Science” can be mapped to {‘Com’, ‘omp’, ‘mpu’, ‘put’, ‘ute’, ‘ter’, ‘er’, ‘r S’, ‘Sc’, ‘Sci’, ‘cie’, ‘ien’, ‘enc’, ‘nce’}. The *q-grams* method is more robust under spelling mistakes and keeps high similarity with different representing orders [6]. The spelling mistake of “Computer” only affect three grams {‘Con’, ‘onp’, ‘npu’}, which take up small parts of the whole string.

Different tokens in the mapping sets may have different ability of discrimination. For example, the token ‘ing’, which appears frequently in words, may have lower significance in discriminating different strings. Tokens are always associated with weights to represent their importance in a string, where text retrieval techniques are commonly used, like Inverse Document Frequency (IDF).

The approaches mentioned above can process entities without many incomplete information formats, but cannot identify the similarity between words and their abbreviation formats. For example, there is even no overlap at all between the q-grams of “Computer Science Department” and its abbreviation “CSD”.

2.2 Matching of Incomplete Formats

We consider two kinds of incomplete information formats, the abbreviation and short words, in our IJoin matching. For the abbreviation, we generate several new elements by connecting the first letters of each word to reserve potential abbreviation information. We also enhance the importance of first few letters in each word by using a decay factor to enlarge the similarity between words and their short formats. The matching steps are described as follows:

Step 1. In order to be robust under different representing orders, we first cut a string into word tokens. Other than *q-grams*, we do not record any information of

word order, such as the connecting token ‘r S’ between two words in the *3-grams* sets of “Computer Science”.

Step 2. We cut each word into *q-grams* to in order to deal with spelling errors. When we search a word in the dictionary, we can find the word with high probability by looking up the first few letters of it. Motivated by this, we associate the first few letters with higher weight than the other ones, by setting a decay factor $\gamma(0 < \gamma \leq 1)$. The weight of *k-th* gram g_k in a word is:

$$w(g_k) = w(g_{k-1}) \cdot \gamma \quad (1)$$

All grams are ordered by the sequence of letters in the word. Note that short word is always few several letters in its original word (e.g. “Conf.” for “Conference”). The decay factor can increase the weight of overlap between the short word and the original one, since we associate higher weight to first few letters (e.g. $w('Con') > w('onf')$ in the word “Conference”).

Step 3. We reconnect the first letters in a string in order to identify the abbreviation of the string. Then, the string of first letters is mapped into *q-grams*. Each gram in this step has equal importance to represent abbreviation, so we do not take decay factor here.

Table 1 shows an example of matching scheme in our IJoin approach. The length of abbreviation word is probably short, so the decay factor does not affect the weight of abbreviation word significantly.

Table 1. Matching scheme in IJoin

<i>String 1, String 2</i>	{Computer Science Department} , {CSD}			
<i>Q-grams of first letters</i>	{CSD,			
<i>Q-grams of each word</i>	g_1	Com ,	Sci ,	Dep ,
	g_2	omp ,	cie ,	ept ,
,	...,	...,
	g_k	ter ,	nce ,	ent }

3 Similarity Function

In this section, we illustrate our similarity function in IJoin. After mapping strings to sets with associated weight of elements, tuples (or entities) can be represented by *vector-space model* [10]. Each tuple t is represented by a vector of weights of p grams (or tokens):

$$t_i = (g_{i1}, \dots, g_{ip}) \quad (2)$$

where t_i is the vector of tuple i and g_{ik} is the weight of gram k in tuple i .

3.1 Cosine Similarity

The similarity between tuples can be quantified by the correlation, ϕ_{ij} , which is so-called the *Cosine Measure*:

$$\phi(t_i, t_j) = \frac{|t_i \cap t_j|}{|t_i \cup t_j|} = \frac{\sum_{k=1}^p g_{ik}g_{jk}}{\sqrt{\sum_{k=1}^p g_{ik}^2 \sum_{k=1}^p g_{jk}^2}} \quad (3)$$

where $\phi(t_i, t_j)$ is the cosine similarity value between tuples t_i and t_j . Clearly, a measure based on cosine similarity can be used when all terms are measured on the same scale. However, cosine similarity is not so effective in dealing with asymmetric features of similarity between words and their incomplete formats. Abbreviation and short word can hardly have high similarity value with their original representing formats.

3.2 IJoin Similarity

Asymmetric features exist in the similarity measure between words and their incomplete formats. Considering the asymmetric features that incomplete formats only take up slight parts of the original information, we calculate the similarity in IJoin as follows:

$$\varphi(t_i, t_j) = \frac{|t_i \cap t_j|}{\min(|t_i|, |t_j|)} = \frac{\sum_{k=1}^p g_{ik}g_{jk}}{\min(\sum_{k=1}^p g_{ik}^2, \sum_{k=1}^p g_{jk}^2)} \quad (4)$$

Other than computing the total weight of two tuples, we use only the smaller one of them. Incomplete formats, such as abbreviation and short word, can keep high similarity with their original formats.

Let us see the previous example in Table 2, where the overlap of two strings' 3-grams is {'CSD'}. We assume that the weight of each gram to be 1. According to the formula (3), the cosine similarity between these two strings is $\phi(string1, string2) = 1/\sqrt{20} = 0.223$. In fact, the cosine similarity value equals to 0 (no overlap), if the IJoin matching is not adopted. For our IJoin similarity function, the similarity value is $\varphi(string1, string2) = 1$.

4 Algorithm Implementation

In this section, we first introduce a basic implementation of our IJoin approach. Then, we discuss a more efficient way to process similarity joins with incomplete information.

4.1 Basic IJoin Implementation

Given two relations R and S , the similarity join operation returns all pairs of tuples r and s from R and S respectively, which satisfy the similarity threshold η (e.g. $\phi(r, s) \geq \eta$). During the preprocessing, we apply the decay factor γ to

each gram of words. *Q-grams* of the words' first letter sequence are added to reserve the potential information of abbreviation.

Finally, we compare each pair of tuples from two relations based on the similarity function in Formula (4). The number of candidates of pairs to be compared is large, especially when the data scale increases. An index of tuples may reduce the accessing time, however, the number of comparisons cannot be decreased. We will discuss a solution to reduce the number of candidates of comparing pairs in the next section.

4.2 Extended IJoin Implementation

In order to reduce the candidates of comparisons, we need to filter those pairs of tuples with no relevancy. A recent study [2] develops a method, namely SSJoin, by exploiting some new attributes for each tuple, which can be used to decide whether two tuples are relevant. The relation $R(A)$ is extended to $R(A, B)$, where B is one of A 's *q-grams*. For example, the string "Computer Science Department" can be extended to 20 tuples as follows:

Table 2. Tuples in SSJoin

R.A String	R.B 3-grams
Computer Science Department	Com
Computer Science Department	omp
Computer Science Department	mpu
...	...
Computer Science Department	men
Computer Science Department	ent

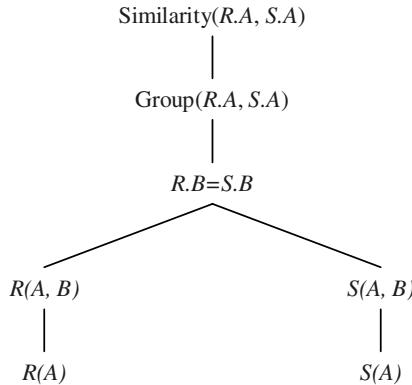
Then, a join operation in SQL (exactly matching) is processed on attribute $R.B$, which finds pairs of tuples with common grams. Those tuples without common grams are totally irrelevant and filtered out. Although the rest pairs of tuples are relevant with common tuples, the number of candidate tuples is still large. The authors use a prefix-filter to reduce duplicate pairs.

In order to improve the efficiency, we do not consider all *q-grams* of A in B like the basic SSJoin. As mentioned before, we can detect a word probably by the first few letters of it. Therefore, in order to improve the efficiency, we consider the first *q-grams* of each word only in the first SQL join operation. This predigest operation will not miss the relevancy between words and their short word formats (e.g. "Conference" and "Conf." have the common gram "Con"). Furthermore, we also add *q-grams* of A 's first letters into B , in order to reserve the potential relevancy between strings and their abbreviation. Table 3 shows an example of tuples in our Extended IJoin approach.

After mapping $R(A)$ and $S(A)$ to $R(A, B)$ and $S(A, B)$ respectively, we operate a SQL join on the attribute B . The result is a group of tuples $(R.A, S.A)$

Table 3. Tuples in IJoin

R.A String	R.B 3-grams
Computer Science Department	Com
Computer Science Department	Sci
Computer Science Department	Dep
Computer Science Department	CSD

**Fig. 1.** Extended IJoins steps

with common grams. Each pair (r, s) in the result can get its similarity value $\varphi(r, s)$ by using the similarity function (4). If the similarity value satisfies the user specified threshold η , (r, s) will be the final join result. Fig 1 shows the process of similarity joins by Extended IJoin.

5 Experimental Evaluation

In this section, we illustrate the results of our experiments which evaluate the effectiveness and efficiency of our IJoin approach. The experiments were performed on a PC, with 2.0GHz CPU and 2GB memory. All programs were implemented in C# and SQL Server.

5.1 Data Sets

We use the real bibliography records from the DBLP web site ¹. The data set consists of all records from 4 conferences, including “VLDB”, “ICDE”, “ICLP” and “FSTTCS”. For each bibliography record, we generated several kinds of citation formats (as shown in Table 4), which are frequently used in bibliography references. We divide all citation records (about 10,000) into two parts, and the

¹ <http://www.informatik.uni-trier.de/~ley/db/>

Table 4. Example of different citation formats

ID	Citations
1	In Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005, 145-156
2	In Proc. of the 31st Int. Conf. on Very Large Data Bases, Trondheim, Norway, Aug 30 - Sep 2, 2005, 145-156
3	In the International Conference on Very Large Data Bases, 2005, 145-156
4	In the Int. Conf. on Very Large Data Bases, 2005, 145-156
5	In the VLDB, 2005, 145-156

similarity joins are performed to find all pairs of citations which represent the same bibliography records.

5.2 Evaluation Criteria

We use the F-Measures with Precision and Recall [8] to evaluate the effectiveness of join operations. Let S_a be actual pairs of citations which represent the same bibliography records, and S_f be pairs found by join operations with high similarity.

$$Recall(S_a, S_f) = \frac{|S_a \cap S_f|}{S_a} \quad (5)$$

$$Precision(S_a, S_f) = \frac{|S_a \cap S_f|}{S_f} \quad (6)$$

$$F(S_a, S_f) = \frac{2 * Recall(S_a, S_f) * Precision(S_a, S_f)}{Recall(S_a, S_f) + Precision(S_a, S_f)} \quad (7)$$

5.3 Effectiveness

In the first experiment, we evaluate the effectiveness by comparing the accuracy of cosine similarity with *q-grams*, basic IJoin and Extended IJoin. We divide 2,000 records of citations into two groups, and process them by different similarity joins approaches. Fig. 2 shows the precision, recall and F-Measure under different specified thresholds of minimum similarity.

Fig. 2 (a) shows that cosine similarity with *q-grams* has a low accuracy in both precision and recall, which means that this approach can hardly find similar entities with incomplete information formats and the obtained results contain many errors. In Fig. 2(b)(c), we can find that our IJoin approach achieves higher precision and recall. When the minimum similarity equals 0.925, it obtains the best balance between precision and recall. We did not apply the decay factor γ to IJoin in this experiment (e.g. $\gamma = 1.0$), which will be evaluated later. And in Fig. 2 (d), we compare F-Measure among these three approaches which

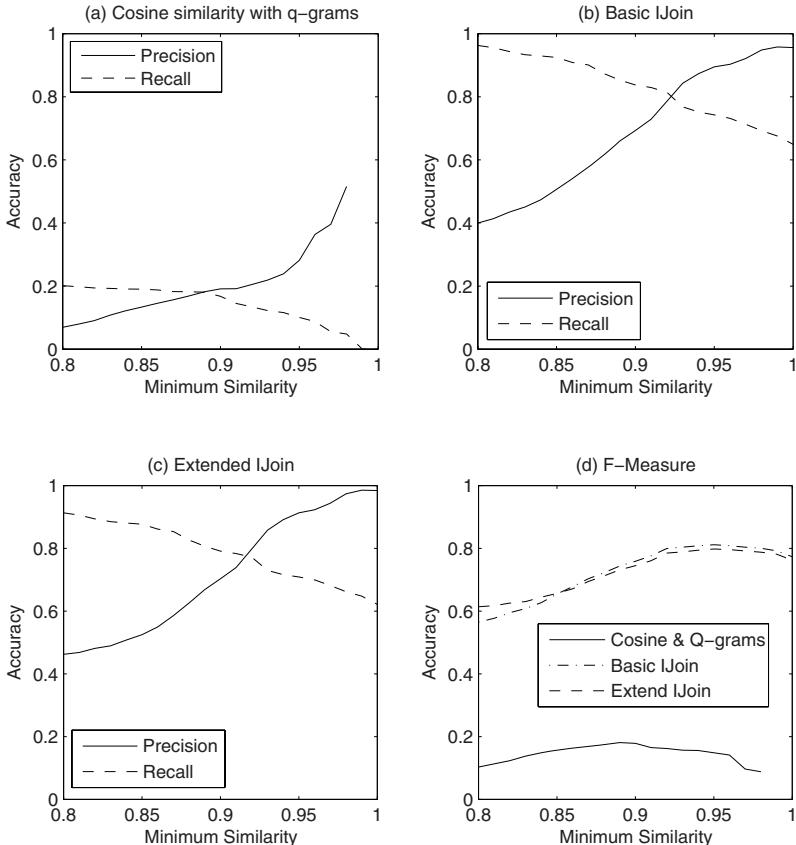
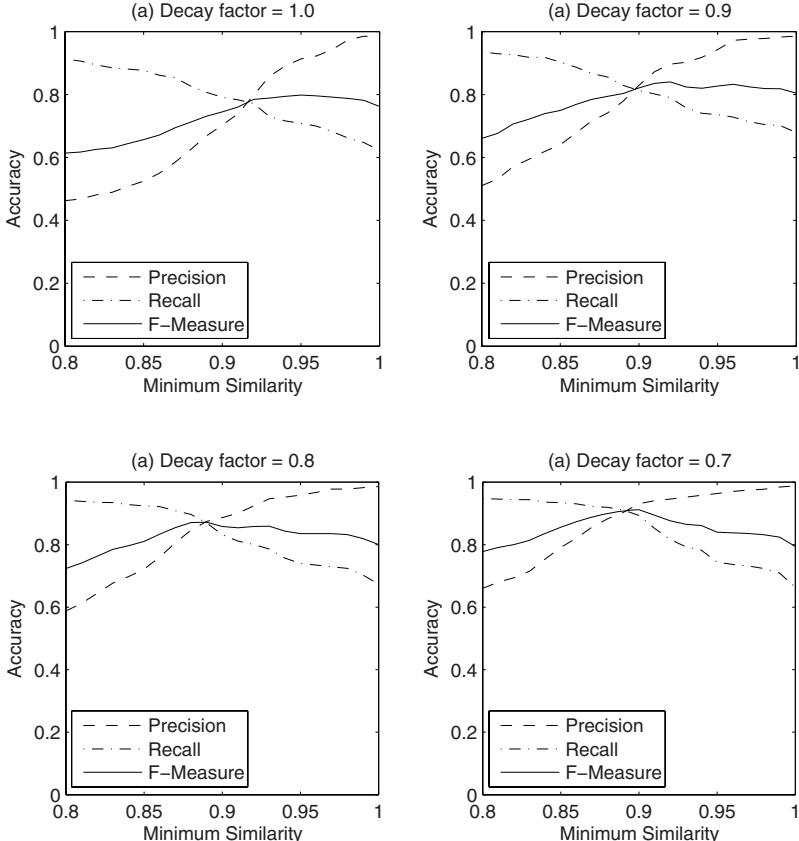


Fig. 2. Accuracy of different approaches

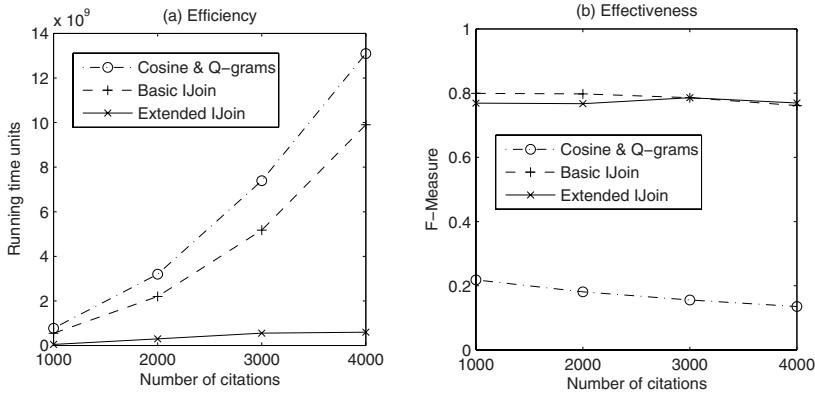
demonstrates the superiority of IJoin in effectiveness. From the figure, we also indicate that Extended IJoin achieves as high accuracy as the basic one, which denotes that the filter operation in Extended IJoin does not influence the effectiveness of IJoin too much. We will further discuss it in the next section.

Then, we evaluate the Extended IJoin with different decay factors to validate that the first several letters have greater importance in a word and can help to improve the accuracy of similarity joins. The experiment is also performed in 2,000 records of citations. Fig. 3 shows the results. With the decrease of decay factor γ , the first few letters (grams) get higher significance in the word which enhances the similarity value between words and their short word formats. As shown in figures, by enhancing the importance of first letters, the accuracy improves when incomplete information formats exist. Note that the decay factor decreases the whole value of all elements in the entity. It is the reason why the best balance point between precision and recall decreases together with the decay factor.

**Fig. 3.** Decay factor in Extended IJoin

5.4 Scalability

In this experiment, we evaluate the scalability of three approaches. Different number of citation records are performed under the best minimum similarity threshold of each approach, e.g. cosine similarity with q-grams (short as cosine & q-grams) achieves the best accuracy at $\eta = 0.89$, basic IJoin at $\eta = 0.925$ and Extended IJoin at $\eta = 0.915$. In order to be comparable with the cosine & q-grams approach, no decay factor is adopted in both IJoin (e.g. $\gamma = 1.0$). Fig. 4 shows the results with different data scales. Fig. 4 (a) illustrates the efficiency of the Extended IJoin approach. Its running time remains low even though the number of citations is multiplied. The time performance of cosine & q-grams and basic IJoin are quite similar and increase exponentially. We also show the accuracy with different data scales in Fig. 4 (b). IJoin approach achieves almost constant accuracy under the same similarity threshold in different numbers of citations. The results confirm the scalability of our Extended IJoin approach in both effectiveness and efficiency.

**Fig. 4.** Different data scales of citations**Table 5.** Basic and Extended IJoin

	Pairs of Comparison	Time Units	Precision	Recall
Basic IJoin	3,998,000	9,901,249	0.714	0.814
Extended IJoin	102,080	596,875	0.748	0.792

Finally, we compare the basic and Extended IJoin in our experiment. Table 5 shows the results in 4,000 records of citations. The filter operation in Extended IJoin reduces the number pairs of comparison greatly by finding out pairs with common elements. The total number of comparisons in basic IJoin is about 40 times greater than that of Extended IJoin. As shown in “Time Units” column, the time performance improved greatly in Extended IJoin. For the effectiveness, although some pairs that are actually similar may be filtered out, the number of such false negatives is not so large and affects the result slightly as shown in the table column “Recall”. Only about 2.2% of similar pairs are missed by Extended IJoin in this experiment. It is interesting that Extended IJoin even achieves higher accuracy than the basic one. This is because most of irrelevant pairs are filtered out and the remaining pairs are probably similar.

6 Conclusions

In this paper, we proposed a novel approach, IJoin, to handle similarity joins of text with incomplete formats, such as abbreviation and short words, are considered in our text matching scheme and similarity function. We connect the first letter of each word to reserve potential abbreviation information and enhance the importance of the first few grams to find high similarity between words and their short formats. The similarity function in IJoin is based on the asymmetric features of similarity between entities with incomplete formats. We also illustrated an efficient implement of our approach (Extended IJoin). Our

experiments showed the advantage of our approach in efficiency and effectiveness when dealing with text entities with incomplete information formats.

References

1. R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.
2. S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5, 2006.
3. W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD Conference*, pages 201–212, 1998.
4. H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.
5. L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.
6. L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. In *WWW*, pages 90–101, 2003.
7. N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD Conference*, pages 802–803, 2006.
8. B. Larsen and C. Aone. Fast and effective text mining using linear-time document clustering. In *KDD*, pages 16–22, 1999.
9. E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *ICDE*, pages 294–301, 1993.
10. G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.
11. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

Self-tuning in Graph-Based Reference Disambiguation*

Rabia Nuray-Turan, Dmitri V. Kalashnikov, and Sharad Mehrotra

Computer Science Department
University of California, Irvine
`{rnruray,dvk,sharad}@ics.uci.edu`

Abstract. Nowadays many data mining/analysis applications use the graph analysis techniques for decision making. Many of these techniques are based on the importance of relationships among the interacting units. A number of models and measures that analyze the relationship importance (link structure) have been proposed (e.g., centrality, importance and page rank) and they are generally based on *intuition*, where the analyst intuitively decides a reasonable model that fits the underlying data. In this paper, we address the problem of *learning such models directly from training data*. Specifically, we study a way to calibrate a *connection strength* measure from training data in the context of *reference disambiguation* problem. Experimental evaluation demonstrates that the proposed model surpasses the best model used for reference disambiguation in the past, leading to better quality of reference disambiguation.

1 Introduction

Many modern data mining and data analysis applications employ decision making capabilities that view the underlying dataset as a graph and then compute the relationship/link importance using various link analysis measures/models including node importance, centrality [29], and page rank [5]. Many of these models are intuition-based and depend on the underlying dataset. In general, since the importance measures are data-driven, a domain analyst decides which measure fits the data best. In the absence of domain analyst, an arbitrary model can be used; however, the results might not be optimal. But, what if there is training data available wherein given any two nodes in the graph it is known which node should be more central/important/etc. Can one design measures that are not purely intuition-based but also take into account such information?

In this paper we provide an answer to that question for one of the graph link analysis measures, called *connection strength* (CS). Given any two nodes u and v in the graph G , the connection strength $c(u, v)$ returns how strongly

* This material is based upon work supported by the National Science Foundation under Award Numbers 0331707 and 0331690. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

u and v are interconnected to each other in G . We study this measure in the context of reference disambiguation problem. In [19,16,17,7,14] a methodology that successfully applies the CS measure to better the disambiguation quality has been proposed.

Reference disambiguation often comes up when entities in a real world dataset contain references to other entities. Frequently, entities are represented using properties/descriptions that may not uniquely identify them leading to ambiguity. For instance, a dataset may store information about two distinct individuals ‘John Smith’ and ‘Jane Smith’, both of whom are referred to as ‘J. Smith’ ambiguously. References may also be uncertain due to differences in the representations of the same entity and errors in data entries (e.g., ‘John Smith’ misspelled as ‘Jon Smith’). The **goal** of reference disambiguation is for each reference to correctly identify the unique entity it refers to.

It is crucial to preprocess and clean the dataset before applying any data mining/analysis applications; because the quality of the output depends on the quality of the input data. Consequently, a large number of database and machine learning approaches have been proposed for solving the reference disambiguation and related disambiguation challenges, such as entity resolution and record linkage [15,11,25,20,28,24,21,18,9,4,14,17,7,16,8].

Recently, some domain-independent data cleaning approaches for reference disambiguation has been proposed [16,22], that systematically exploits features and relationships among entities for the purpose of disambiguation. The approach in [16], which we employ to test our adaptive solution, views the dataset as a graph of entities that are linked to each other via relationships. The model first utilizes a feature based method to identify a set of candidate entities for a reference. Graph theoretic techniques are then used to discover and analyze relationships that exist between the entity containing the reference and the set of candidates. The analysis is based on the CAP principle:

Context Attraction Principle(CAP): *If reference r made in the context of entity x refers to and entity y_j , whereas the description provided by r matches multiple entities y_1, y_2, \dots, y_N , then x and y_j are likely to be more strongly connected to each other via chains of relationships than x and y_ℓ ($\ell = 1, 2, \dots, N; \ell \neq j$)*

To illustrate the CAP, consider a simple publication scenario, where ‘authors’ write ‘papers’, and ‘authors’ are affiliated with some ‘organizations’. For instance, some paper P_1 might mention ‘J. Smith’ as its author. The dataset might contain only two people who have similar names: John and Jane Smith. Then $r = ‘J. Smith’$, $x = P_1$, $y_1 = ‘John Smith’$, and $y_2 = ‘Jane Smith’$. To decide if the ‘J. Smith’ is Jane or John, the CAP proposes to compare two sets of paths in the entity-relationship graph that exist between x and y_1 and between x and y_2 .

The **main contribution** of this paper is a supervised learning algorithm that learns the importance of relationships, or CS, among the classified entities and makes the approach self-tunable to any underlying domain so that the participation of the domain analyst is minimized significantly.

The rest of this paper is organized as follows. Section 2 covers related work. Section 3 defines the problem of reference disambiguation and the essence of the disambiguation approach we use. An adaptive model for CS is discussed in Section 4. The empirical evaluation of the proposed solution is covered in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

In this section we give a brief overview of the existing connection strength models (Section 2.1) and the reference disambiguation techniques (Section 2.2).

2.1 Connection Strength Models

The connection strength $c(u, v)$ between two nodes u and v reflects how strongly these nodes are related to each other via relationships in the graph. Generally, a domain expert decides a mathematical model to compute $c(u, v)$, which describes the underlying dataset best. Various research communities have proposed measures that are directly related to $c(u, v)$. Below we summarize some of the principal models.

Diffusion kernels on graphs in kernel methodology [26] is directly related to connection strength. Diffusion kernel methods view the underlying dataset as a graph $G = (V, E)$, where V is a set of entities and E is a set of edges which define the base similarities between entities. The base similarity for entities $x, y \in V$ represents the degree of attraction between x and y . Moreover, the base similarities are used to compute indirect similarities by combining the direct similarities in a particular way, see [26] for details.

Another model of measuring CS is *random walks* in graphs. It has been extensively studied, including our previous work [17, 16]. The model uses the probability of reaching node v from node u by random walks in G to compute $c(u, v)$. *Relevant importance in graphs* [30] is a generalized version of page rank algorithm [5]. It studies the relevant importance of a set of nodes with respect to a set of root nodes. The connection strength in this study is the importance of node t given node r (i.e., $I(t|r)$). *Electric circuit model* is also a CS model which uses the electric networks principles to find the connection subgraphs between two nodes u and v [10]. That model views the graph as an electric circuit consisting of resistors, and compute $c(u, v)$ as the amount of electric current that goes from u to v .

2.2 Disambiguation

Reference disambiguation problem is related to the *record de-duplication*, *record linkage*, and *object consolidation* problems [7, 21] and often arises when different information sources are merged to create a single database. The differences between record linkage and reference disambiguation can be intuitively viewed using the relational terminology as follows: while the record linkage problem

consists of determining when two records are the same, reference disambiguation corresponds to ensuring that references in a database point to the correct entities. In the reference disambiguation problem, for each reference, a set of possible candidates is given and the task is to pinpoint the correct entity in this set. On the other hand, the object consolidation problem aims to correctly group/cluster the references that refer to the same object without knowing the possible entities in the dataset.

The traditional approach to these problems is to analyze the textual similarities among the object features to make a disambiguation decision. Such approaches are called feature-based similarity (FBS) techniques [23, 11, 12]. Recently, a number of techniques have been proposed that go beyond the traditional approach [14, 11, 3, 24, 27, 21, 9, 17, 16, 7, 2, 22]. Ananthakrishna et al [1] presented relational deduplication in data warehouses where there is dimensional hierarchy over the relations. Bhattacharya and Getoor introduced a method which requires that social groups function as cliques [3]. This model expects that there are strong correlations between pairs or sets of entities, such that they often co-occur in information sources. Bekkerman and McCallum studied the disambiguation of name references in a linked environment [2]. Their model utilizes the hyperlinks and distance between the pages where ambiguous names are found. Minkov et al [22] introduced extended similarity metrics for documents and other objects embedded in graphs, facilitated by a lazy graph walk. They also introduced a learning algorithm which adjusts the ranking of possible candidates based on the edges in the paths traversed.

In this paper, we employ the algorithm presented in [17, 16] to test our adaptive connection strength model. The algorithm uses a graphical methodology; the disambiguation decisions are made not only based on object features like in the traditional approach, but also based on the inter-object relationships, including indirect ones that exist among objects. The essence of the adaptive model is to be able to learn the importance of various connections on past data in the context of reference disambiguation.

3 Problem Definition

We now formally define the reference disambiguation problem. Assume dataset \mathcal{D} contains a set of entities X . Each entity $x \in X$ itself consists of one or more attributes $x.a_1, x.a_2, \dots$, and it might also contain several references $x.r_1, x.r_2, \dots$ to other entities in X . Let R be the set of all references. Each reference $r \in R$ is essentially a description and may itself contain one or more attributes. For each reference $r \in R$ the **option set** S_r of that reference is known. It contains all entities in X to which r might potentially refer: $S_r = \{y_{r1}, y_{r2}, \dots, y_{rn_r}\}$. For r its S_r is initially determined either by ad hoc techniques, domain knowledge, or by choosing all entities whose feature-based similarity exceed a certain threshold. The true (unknown to the algorithm) entity to which r refers to is denoted as r^* . Then the goal of reference disambiguation is to pick the right y_{rj} (i.e., r^*) from S_r to which r really refers to.

We denote the entity in the context of which reference r is made as x_r . The employed reference disambiguation approach resolves each reference $r \in R$ by analyzing direct and indirect relationships that exist between x_r and each member of S_r . For that, it views the dataset \mathcal{D} as an undirected entity-relationship graph G , where nodes represent entities and edges represent relationships. In essence, G can be viewed as an instantiation of the E/R diagram for \mathcal{D} . The approach relies on the CAP principle (Section II), which can be reformulated in terms of connection strength as: for r its r^* is likely to be such element y_{rj} from S_r that $c(x_r, y_{rj}) \geq c(x_r, y_{r\ell})$ for all $\ell \neq j$.

To make the definition clear, let us assume that we use the publication dataset for reference disambiguation. In the publication domain ‘authors’ write ‘papers’. We might have a paper P_1 that mentions ‘J. Smith’ as its author. Dataset \mathcal{D} might contain two authors who match that description: John Smith and Jane Smith, where the actual author of P_1 is John. Then $r = ‘J. Smith’$, $x_r = P_1$, $r^* = ‘John Smith’$, and $S_r = \{‘John Smith’, ‘Jane Smith’\}$.

4 Solution

The core of the approach in [17,16] that we employ to test our adaptive solution is a connection strength model, called WM. It is a *fixed* mathematical model and based on some intuitive assumptions which are true for many datasets. In this section we first describe how an *adaptive* CS model can be created (Section 4.1). Then we give an example adaptive CS model (Section 4.2) which is used in this paper. Finally, we discuss the self-tuning algorithm (Section 4.3).

4.1 Adaptive Connection Strength Model

Assume that we can classify each path that the disambiguation algorithm finds in graph G into a finite set of path types $S_T = \{T_1, T_2, \dots, T_n\}$. Namely, there is a function $T(p, G) \rightarrow S_T$ such that for any given path p and graph G , maps it to one of those path types. If any two paths p_1 and p_2 are of the same type T_j , then they are treated as identical by the algorithm. Then, for any two nodes u and v we can characterize the connections among them with a path-type count vector $T_{uv} = (c_1, c_2, \dots, c_n)$, where each c_i is the number of paths of type T_i that exist between u and v . If we assume that there is a way to associate weight w_i with each path type T_i , then CS model computes $c(u, v)$ as:

$$c(u, v) = \sum_{i=1}^n c_i w_i. \quad (1)$$

The existing CS models differ in classification of path types and in the way of assigning weights to path types. Furthermore, these are generally chosen by the algorithm designer.

4.2 Path Type Model

To classify the paths we use a model which we refer to as Path Type Model (PTM). It classifies paths by looking at the types of edges the path is comprised of. Namely, PTM views each path as a sequence of edges (e_1, e_2, \dots, e_k) , where each edge has a type associated with it. This sequence of edge types $((E_1, E_2, \dots, E_k))$ are treated as a string and PTM assigns different weights to each string. For example, in the publications domain authors write papers and are affiliated with organizations. Hence there are two types of edges that correspond to the two types of relationships: E_1 for ‘writes’ and E_2 for ‘is affiliated with’.

4.3 Learning Algorithm

The CAP principle in the reference disambiguation problem allows us to calibrate a CS model directly from data and apply it in the context of reference disambiguation. The principle states that for a reference r it is likely that

$$c(x_r, y_{rj}) \geq c(x_r, y_{r\ell}) \text{ for any } r, \ell \neq j \text{ where } y_{rj} = r^*. \quad (2)$$

Because of the ‘likely’ part in the CAP, many of the inequalities in the system (2) should hold, but some of them might not. That is, system (2) might be overconstrained and might not have a solution. To address the ‘likely’ part, we add a slack to the system and then require it be minimized:

$$\left\{ \begin{array}{l} \text{Constraints:} \\ c(x_r, y_{rj}) + \xi_{r\ell} \geq c(x_r, y_{r\ell}) \text{ for any } r, \ell \neq j, y_{rj} = r^* \\ \xi_{r\ell} \geq 0 \\ \text{Objective:} \\ \text{Minimize } \sum_{r\ell} \xi_{r\ell}. \end{array} \right. \quad (3)$$

The employed reference disambiguation approach also states that for reference r the connection strength ‘evidence’ for the right option $y_{rj} = r^*$ should visibly outweigh that for the wrong ones $y_{r\ell}, \ell \neq j$. Thus, in addition to the objective in (3), the value of $[c(x_r, y_{rj}) - c(x_r, y_{r\ell})]$ should be maximized for all $r, \ell \neq j$, which translates into maximizing $\sum_{r\ell} [c(x_r, y_{rj}) - c(x_r, y_{r\ell})]$. After combining the first and second objectives, we have:

$$\left\{ \begin{array}{l} \text{Constraints:} \\ c(x_r, y_{rj}) + \xi_{r\ell} \geq c(x_r, y_{r\ell}) \text{ for any } r, \ell \neq j, y_{rj} = r^* \\ \xi_{r\ell} \geq 0 \\ \text{Objective:} \\ \text{Minimize } \alpha \sum_{r\ell} \xi_{r\ell} + (1 - \alpha) \sum_{r\ell} [c(x_r, y_{rj}) - c(x_r, y_{r\ell})] \end{array} \right. \quad (4)$$

Here α is a parameter that allows to vary the contribution of the two different objectives. It is a real number between 0 and 1, whose optimal value can be

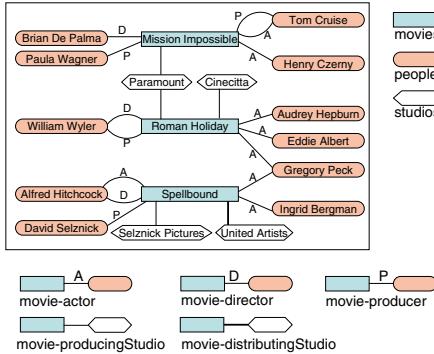


Fig. 1. Movies Dataset: Sample entity-relationship graph

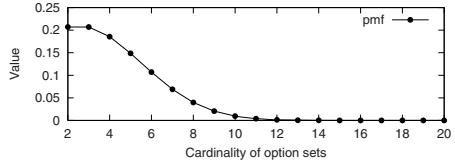


Fig. 2. PMF of sizes of option sets

determined by varying α on training data and observing the effect on the quality of the disambiguation. System (II) essentially converts the learning task into solving the corresponding linear programming problem, and linear programming, in general, is known to have efficient solutions [13]. All $c(u, v)$ in (II) should be computed according to (II) and adding a normalization constraint that all weights should be in $[0, 1]$ domain: $0 \leq w_i \leq 1$, for all i . The task becomes to compute the best combination of weights w_1, w_2, \dots, w_n that minimizes the objective, e.g. using any off-the-shelf linear programming solver.

5 Experimental Results

We experimentally study our method using real and synthetic datasets taken from two domains: Movies (Section 5.1) and Publications (Section 5.2). We compare the learning approach (PTM) against the best existing model used for disambiguation so far: the random walk model (WM) [17], which we will refer to as RandomWalk.

RandomWalk model computes $c(u, v)$ as the probability to reach node v from node u via random walks in graph G , such that the probability to follow an edge is proportional to the weight of the edge. Accordingly, $c(u, v)$ is computed as the sum of the connection strength $c(p)$ of each path p from $P_L(u, v)$, where $c(p)$ is the probability of following path p in G , i.e.

$$c(u, v) = \sum_{p \in P_L(u, v)}^n c(p) \quad (5)$$

We report the results in terms of **accuracy**¹, which is defined as the fraction of correctly resolved references.

¹ For the reference disambiguation problem we solve, the accuracy and F-measure are known to be the same.

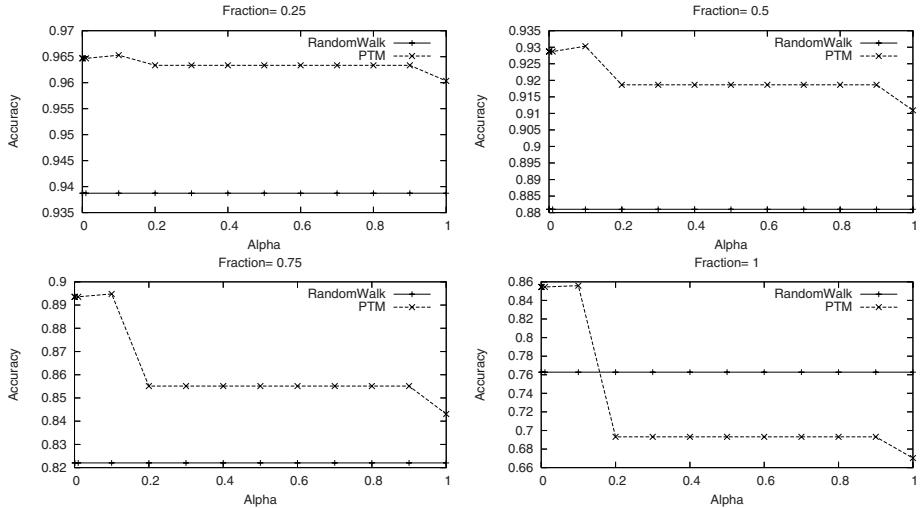


Fig. 3. Accuracy vs. Alpha: where $c = 2$

5.1 Experiments on the Movies Domain

We use the Stanford Movies Dataset². A sample entity-relationship graph for this dataset is illustrated in Figure 1. The dataset contains three different entity types: *movies* (11,453 entities), *studios* (992 entities) and *people* (22,121 entities) and there are five types of relationships: *actors*, *directors*, *producers*, *producing studios* and *distributing studios*.

When studying the accuracy of disambiguation, we use a method of testing commonly employed by many practitioners, including the recent KDD CUP. We introduce uncertainty in the dataset manually in a controlled fashion and then analyze the resulting accuracy of various methods. Specifically, we disambiguate references from movies to directors, by making them uncertain. First, a fraction $f : 0 \leq f \leq 1$ of all director references is chosen to be made uncertain, while the rest remain certain. Each to-be-uncertain director reference r is made ambiguous by modifying it such that it either points to two directors instead of one (i.e., $c = |S_r| = 2$) or points to c directors where c is distributed according to the PMF in Figure 2 (i.e., $c = |S_r| \sim pmf$). Here c stands for the **cardinality** of S_r . Training and testing is performed for the same values of f and c , but the director references chosen to be ambiguous are different in training and test data.

Figures 3 and 4 study the effect of the parameter α , that controls the contribution of various objectives in system (4) from Section 4.3, on the accuracy of various approaches, for different combinations of f and c . We performed the experiments for two different cardinalities ($c = 2$ and $c \sim pmf$) and four different fractions of ambiguous entities ($f = \{0.25, 0.5, 0.75, 1\}$). In these experiments, the optimal α was found to be approximately 0.10. When α is set to its best

² <http://www-db.stanford.edu/pub/movies/>

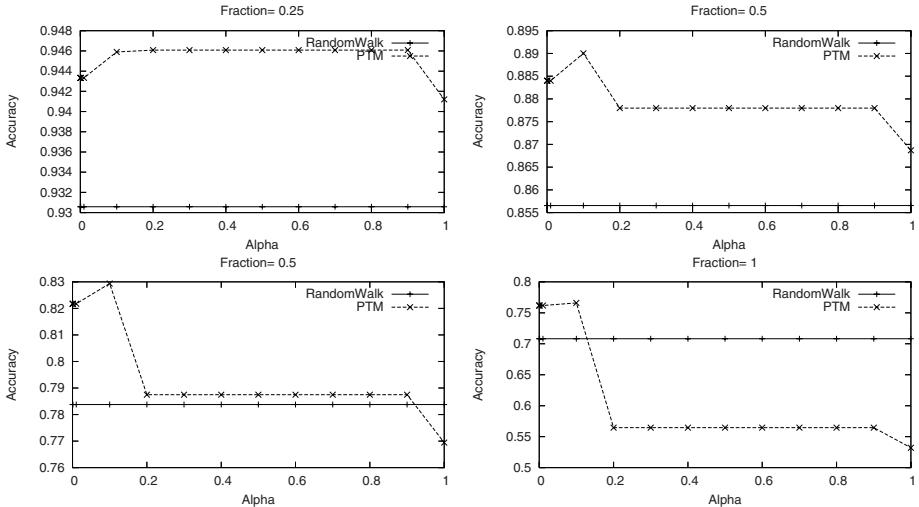


Fig. 4. Accuracy vs. Alpha: where $c \sim pmf$

value, PTM visibly outperforms the state-of-the-art model, RandomWalk. As the number of ambiguous references increases, the improvement with the PTM against the RandomWalk method becomes more significant. For example, the improvement with PTM for $f = 0.25$ and $c = 2$ is 2.7%, whereas it is 9.18%, when $f = 1$ and $c = 2$.

5.2 Experiments on the Publications Domain

Dataset. We now present the results on **SynPub** dataset, which is from [16] and emulates CiteSeer dataset. It contains four different types of entities: *author*, *paper*, *department*, and *organization* and three types of relationships: *author-paper*, *author-department*, and *department-organization*.

We generated *five* different sets of datasets. Each set contains a training and *ten* different testing datasets, the parameters are same for all datasets; however, the authors to be disambiguated are different. Each dataset has different types and levels of uncertainty (see [17]) and contains 5000 papers, 1000 authors, 25 organizations, and 125 departments. The least ambiguous datasets are in set 4, while the most ambiguous ones are either in set 5 or set 1, see Table II.

Results. For each training dataset, we selected the optimal α value, which is 0.10 for datasets 1, 2, and 5 and 0.01 for dataset 3 and 4. Then these optimal values were used in testing. The average accuracy of different testings are reported in Table II. Since the results of PTM and RandomWalk are essentially identical, we performed another experiment with a different path classification model, **hybrid** model. This model is the combination of PTM with RandomWalk, such that it takes into account the node degrees in addition to the edge types in a path. The connection strength of this model is computed as:

$$c(u, v) = \sum_{p \in P_L(u, v)}^n c(p) w_{T_i}, \text{ where } T_i = T(p, G) \quad (6)$$

Accuracy results with the hybrid model is the same as the other two models. So we can conclude that RandomWalk model is a good model for this specific setting. However, it may not work ideally for every instance of the publications domain. To show that, we performed some additional experiments. Our intuition in these experiments is that when creating the SynPub dataset, the analyst has chosen to project from CiteSeer relationships of only a few carefully chosen types that would work well with RandomWalk, i.e. the three types discussed above, while purposefully pruning away relationship types that are less important for disambiguation and would confuse RandomWalk model. In other words, the analyst has contributed his intelligence to that unintelligent model.

We gradually added random noise to one of the datasets, namely dataset 5, by introducing relationships of a new type – that represent random meaningless relationships. The random relationships were added to the ‘false’ cases only. That is, the added relationships are between the reference r and the candidates $(y_{rj}) \in \{S_r - r^*\}$. Figure 5 examines the effect of this noise on the accuracy of RandomWalk and PTM techniques. It shows that both of the techniques obtain very high accuracy compared to the standard approach, shown as ‘FBS’, which does not use relationships for disambiguation. Initially, RandomWalk and

Table 1. Publications dataset results

Dataset	PTM	RandomWalk	Hybrid	FBS
1	93.2%	93.1%	93.1%	50.0%
2	94.9%	94.9%	94.9%	55.0%
3	74.6%	74.7%	74.7%	55.0%
4	98.4%	98.4%	98.4%	74.8%
5	64.9%	64.9%	64.9%	50.0%

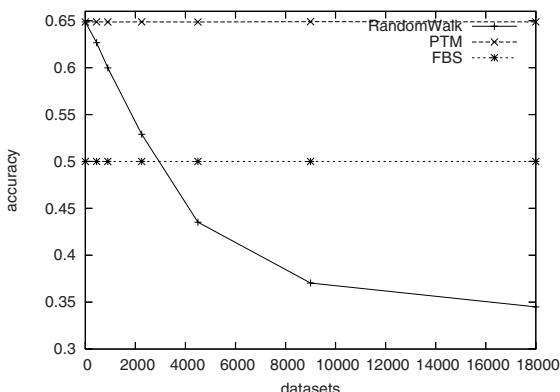


Fig. 5. Accuracy vs. Number of random relationships(noise)

PTM has the same accuracy. But as the level of noise increases, the accuracy of RandomWalk drastically drops below that of PTM and FBS. PTM is an intelligent technique that learns the importance of various relationships and can easily handle noise – its curve stays virtually flat. Notice, since FBS does not use any relationships, including the random noise, its curve stays flat as well.

6 Discussions and Conclusion

Our results show that adaptive connection strength model always outperforms the state-of-the-art RandomWalk model. There are many advantages of self-tunable CS model in the context of reference disambiguation. First of all, it minimizes the analyst participation, which is important since nowadays various data-integration solutions are incorporated in real Database Management Systems (DBMS), such as Microsoft SQL Server DBMS [6]. Having a less analyst-dependent technique makes that operation of wide applicability, so that non-expert users can apply it to their datasets. The second advantage of such a CS model is that it expects to increase the quality of the disambiguation technique. There are also less obvious advantages. For example, the technique is able to detect which path types are marginal in their importance. Thus, the algorithm that discovers paths when computing $c(u, v)$ can be sped up, since the path search space can be reduced by searching only for important paths. Speeding up the algorithm that discovers paths is important since it is the bottleneck of the overall disambiguation approach [17,15,16].

References

1. R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, 2002.
2. R. Bekkerman and A. McCallum. Disambiguating web appearances of people in a social network. In *WWW*, 2005.
3. I. Bhattacharya and L. Getoor. Relational clustering for multi-type entity resolution. In *MRDM Workshop*, 2005.
4. M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *SIGKDD*, 2003.
5. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proc of International World Wide Web Conference*, 1998.
6. S. Chaudhuri, K. Ganjam, V. Ganti, R. Kapoor, V. Narasayya, and T. Vassilakis. Data cleaning in Microsoft SQL Server 2005. In *SIGMOD*, 2005.
7. Z. Chen, D. V. Kalashnikov, and S. Mehrotra. Exploiting relationships for object consolidation. In *Proc. of International ACM SIGMOD Workshop on Information Quality in Information Systems (ACM IQIS 2005)*, Baltimore, MD, USA, June 17 2005.
8. W. Cohen, H. Kautz, and D. McAllester. Hardening soft information sources. In *SIGKDD*, 2000.
9. X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, 2005.

10. C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *SIGKDD*, 2004.
11. I. Fellegi and A. Sunter. A theory for record linkage. *Journal of Amer. Statistical Association*, 64(328):1183–1210, 1969.
12. M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, 1995.
13. F. Hillier and G. Lieberman. *Introduction to operations research*. McGraw-Hill, 2001.
14. D. V. Kalashnikov, S. Chen, R. Nuray-Turan, S. Mehrotra, and N. Ashish. Disambiguation algorithm for people search on the web. In *Proc. of the IEEE 23rd International Conference on Data Engineering (IEEE ICDE 2007)*, Istanbul, Turkey, April 16–20 2007.
15. D. V. Kalashnikov and S. Mehrotra. RelDC project.
<http://www.ics.uci.edu/~dvk/RelDC>
16. D. V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems (ACM TODS)*, 31(2):716–767, June 2006.
17. D. V. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM International Conference on Data Mining (SIAM Data Mining 2005)*, Newport Beach, CA, USA, April 21–23 2005.
18. X. Li, P. Morie, and D. Roth. Identification and tracing of ambiguous names: Discriminative and generative approaches. In *AAAI*, 2004.
19. B. Malin. Unsupervised name disambiguation via social network similarity. In *Workshop on Link Analysis, Counterterrorism, and Security*, 2005.
20. A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *ACM SIGKDD*, 2000.
21. A. McCallum and B. Wellner. Object consolidation by graph partitioning with a conditionally-trained distance metric. In *KDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*, 2003.
22. E. Minkov, W. W. Cohen, and A. Ng. Contextual search and name disambiguation in email using graphs. In *SIGIR*, 2006.
23. H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, 1959.
24. H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In *NIPS Conference*, 2002.
25. S. Sarawagi and A. Bhagat. Interactive deduplication using active learning. In *SIGKDD*, 2002.
26. J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
27. P. Singla and P. Domingos. Multi-relational record linkage. In *MRDM Workshop*, 2004.
28. S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *SIGKDD*, 2002.
29. S. Wasserman and K. Faust. *Social Network Analysis Methods and Applications*. Cambridge University Press, 1994.
30. S. White and P. Smyth. Algorithms for estimating relative importance in networks. In *SIGKDD*, 2003.

Probabilistic Nearest-Neighbor Query on Uncertain Objects

Hans-Peter Kriegel, Peter Kunath, and Matthias Renz

University of Munich, Germany

{kriegel, kunath, renz}@dbs.ifai.lmu.de

Abstract. Nearest-neighbor queries are an important query type for commonly used feature databases. In many different application areas, e.g. sensor databases, location based services or face recognition systems, distances between objects have to be computed based on vague and uncertain data. A successful approach is to express the distance between two uncertain objects by probability density functions which assign a probability value to each possible distance value. By integrating the complete probabilistic distance function as a whole directly into the query algorithm, the full information provided by these functions is exploited. The result of such a probabilistic query algorithm consists of tuples containing the result object and a probability value indicating the likelihood that the object satisfies the query predicate. In this paper we introduce an efficient strategy for processing probabilistic nearest-neighbor queries, as the computation of these probability values is very expensive. In a detailed experimental evaluation, we demonstrate the benefits of our probabilistic query approach. The experiments show that we can achieve high quality query results with rather low computational cost.

1 Introduction

In many modern application ranges, e.g. spatio-temporal query processing of moving objects [4], sensor databases [3] or personal identification systems [13], usually only uncertain data is available. In the area of multimedia databases, e.g. image or music databases, or in the area of personal identification systems based on face recognition and fingerprint analysis, there often exists the problem that a feature vector cannot exactly be determined. This “positional” uncertain data can be handled by assigning confidence intervals to the feature values, by specifying probability density functions indicating the likelihood of certain feature values, or by specifying confidence values for a set of discrete feature values. The advantage of the latter form of representation of uncertain data is that distances between the uncertain objects can be processed more easily than object distances based on smooth probability density functions. Furthermore, positional uncertainties of objects are often given in form of discrete values, in particular, if potential object locations are derived from different observations. Even when the uncertainty of the objects are specified by means of smooth probability density functions, we can achieve our preferred discrete data representation by means of sampling techniques. With this concept, we can find a good trade-off between accuracy and query performance.

The approach proposed in [9] which uses probabilistic distance functions to measure the similarity between uncertain objects seems very promising for probabilistic similarity queries, in particular for the probabilistic distance-range join. Contrary to traditional

approaches, they do not extract aggregated values from the probabilistic distance functions but enhance the join algorithms so that they can exploit the full information provided by these functions. The resulting probabilistic similarity join assigns a probability value to each object pair indicating the likelihood that the pair belongs to the result set, i.e. these probably values reflect the trustability of the result. In this paper, we adopt the idea to use probabilistic distance functions between positional uncertain objects in order to assign probability values to query results reflecting the trustability of the result. In applications where wrong results have fatal consequences, e.g. medical treatment, users might only look at very certain results, whereas in commercial advertising, for instance, all results might be interesting. Based on this concept, we propose a solution for probabilistic nearest neighbor queries which are practically very important in many application areas.

2 Related Work

In the last decade, a lot of work has been done in the field of similarity query processing with the focus on management and processing of uncertain data. Thereby, the development of efficient and effective approaches providing probabilistic query results were of main interest. A survey of the research area concerning uncertainty and incomplete information in databases is given in [1] and [11]. Recently a lot of work has been published in the area of management and query processing of uncertain data in sensor databases [3] and especially in moving object environments [4, 12]. Similar to the approach presented in this paper, the approaches in [2, 3, 4, 12] model uncertain data by means of probabilistic density functions (*pdfs*). In [12], for instance, moving objects send their new positions to the server, iff their new positions considerably vary from their last sent positions. Thus, the server always knows that an object can only be a certain threshold value away from the last sent position. The server, then, assigns a *pdf* to each object reflecting the likelihood of the objects possible positions. Based on this information the server performs probabilistic range queries. Likewise, in [4] an approach is presented for probabilistic nearest neighbor queries. Note that both approaches assume non-uncertain query objects, and thus, they cannot be used for queries where both query and database objects are uncertain. Queries that support uncertain database objects as well as uncertain query objects are very important as they build a foundation for probabilistic join procedures. Most recently, in [9] a probabilistic distance range join on uncertain objects was proposed. Instead of applying their join computations directly on the *pdfs* describing the uncertain objects, they used sample points as uncertain object descriptions for the computation of the probabilistic join results.

Furthermore, most recently [5] an approach was proposed dealing with spatial query processing not on positionally uncertain data but on existentially uncertain data. This kind of data naturally occurs, if, for instance, objects are extracted from uncertain satellite images. The approach presented in this paper does not deal with existentially uncertain data but with positionally uncertain data which can be modelled by probability density functions or are already given as probabilistic set of discrete object positions similar to the approach presented in [9].

3 Probabilistic Nearest Neighbor Query on Uncertain Data

As already mentioned, a non-probabilistic similarity query on positional uncertain data has some limitations which are overcome by our probabilistic approach introduced in this section. It is based on a direct integration of the probabilistic distance functions rather than using only aggregated values. Our new query type assigns to each result object a probability value reflecting the likelihood that the object fulfills the query predicate.

Definition 1 (probabilistic similarity query)

Let q be an uncertain object and DB denote a database, and let θ_d denote any similarity query predicate based on a given distance function d . Furthermore, let $P(q \theta_d o)$ denote the probability that $q \theta_d o$ is true for the object pair $(q, o) \in q \times DB$. Then, the *probabilistic similarity query* Q_{θ}^{prob} consists of result pairs $(o, P(q \theta_d o)) \in DB \times [0,1]$ for which $P(q \theta_d o) > 0$ holds, i.e. $Q_{\theta}^{\text{prob}} = \{(o, P(q \theta_d o)) \mid P(q \theta_d o) > 0\} \subseteq DB \times [0,1]$.

3.1 Probabilistic Nearest-Neighbor Query Based on Smooth Probabilistic Distance Functions

In this section, we shortly show how we can theoretically compute the probability value $P(q \theta_d^{nn} o)$ underlying the probabilistic nearest-neighbor query.

Lemma 1. For a given uncertain query object q each uncertain database object o , we can compute $P(q \theta_d^{nn} o)$ reflecting the probability that o is the nearest neighbor of q as follows:

$$P(q \theta_d^{nn} o) = \iint_{I^d} q(v) \cdot \left[\int_{-\infty}^{+\infty} f_d(\delta(\tau - v), o)(\tau) \cdot \prod_{x \in DB \setminus o} \left(1 - \int_{-\infty}^{\tau} f_d(\delta(\tau - v), x)(t) dt \right) d\tau \right] dv$$

Proof. First, we fix a certain position v for the uncertain object representation q . Then, we weigh the probabilistic distance function $f_d(\delta(\tau - v), o)(\tau)$ between our uncertain object o and our “certain” position v with a probability value P_{weight} indicating the likelihood that all database objects $x \in DB \setminus o$ have a distance higher than τ from v . Integrating, over all distance values τ yields the probability that o is the nearest neighbor of q under the condition that the position of q is equal to v . Finally, integrating over all possible positions of q yields the probability that o is the nearest neighbor of q . \square

Note that we can extend Lemma 1 so that it can be used as foundation for the probabilistic nearest-neighbor query, by substituting the probability value P_{weight} by the following expression:

$$\sum_{\substack{A \subseteq DB \setminus o \\ |A| = k-1}} \prod_{y \in A} \left(\int_{-\infty}^{\tau} f_d(\delta(\tau - v), y)(t) dt \right) \cdot \left(1 - \int_{-\infty}^{\tau} f_d(\delta(\tau - v), x)(t) dt \right)$$

3.2 Probabilistic Nearest-Neighbor Query Based on Discrete Probabilistic Distance Representations

Although for some uncertain object representations it would be possible to compute the probabilistic similarity queries directly on Lemma 1, we propose to compute them

based on the generally applicable concept of monte-carlo sampling. In many applications the uncertain objects might already be described by a discrete probability density function, i.e. we have the sample set already. If the uncertain object is described by a continuous probability density function, we can easily sample according to this function and derive a set of samples. In the following, we assume that each object o is represented by a set of s sample points, i.e. o is represented by s different representations $\{o_1, \dots, o_s\}$. After having described how to organize these discrete object representations within a database (cf. Section 3.2.1), we show how to compute the probabilistic nearest-neighbor query (cf. Section 3.2.2) based on these discrete object representations.

3.2.1 Database Integration of Uncertain Data

In order to reduce the complexity of the query computation, we introduce an efficient query algorithm which is based on groups of samples. Thereby two samples o_i and o_j of the same object o are grouped together to one cluster, if they are close to each other. We can generate such a clustering on the object samples by applying the partitioning clustering algorithm k -means [10] individually to each sample set $\{o_1, \dots, o_s\}$. Thus, an object is no longer approximated by s samples, but by k clusters containing all the s sample points of the object.

Definition 2 (clustered object representation)

Let $\{o_1, \dots, o_s\}$ be a discrete object representation. Then, we call the set $\{\{o_{1,1}, \dots, o_{1,n_1}\}, \dots, \{o_{k,1}, \dots, o_{k,n_k}\}\}$ a *clustered object representation* where $\bigcup_{i=1 \dots k, j=1 \dots n_i} o_{i,j} = \{o_1, \dots, o_s\}$ and $n_1 + \dots + n_k = s$.

Similar to [9], we store these clustered object representations in R-tree [6] like index structures.

3.2.2 Nearest-Neighbor Query Algorithm

A straightforward approach for an efficient probabilistic nearest-neighbor query is based on the *minimal maximum distance* d_{minmax} (cf. Definition 3). Based on this distance it is possible to exclude many database objects o from the probabilistic nearest-neighbor search of a query object q (cf. Lemma 2).

Definition 3 (minimal maximum object distance)

Let q be an uncertain query object. Then the *minimal maximum object distance* of q is computed by: $d_{minmax} = \min \{maxdist(\text{MBR}(q), \text{MBR}(o)) | o \in DB\}$

Lemma 2. Let q be an uncertain query object and DB be a set of uncertain objects. Then, the following statement holds:

$$\forall o \in DB: mindist(\text{MBR}(q), \text{MBR}(o)) > d_{minmax} \Rightarrow P(q \theta_d^{mn} o) = 0$$

Proof. Let $o' \in DB$ be the object in the database for which $maxdist(\text{MBR}(q), \text{MBR}(o')) = d_{minmax}$ holds. Then, for all sample points $q_{i,j}, o_{i,j}, o'_{i',j'}$ the following statement holds: $d(q_{i,j}, o_{i,j}) > d(q_{i,j}, o'_{i',j'})$. Therefore, the probability that o is the nearest neighbor of q is equal to 0. \square

Based on the candidate sets $C = \{o \in DB | mindist(\text{MBR}(q), \text{MBR}(o)) \leq d_{minmax}\}$, we can introduce a straightforward approach which computes the probability value $p_{nn}(q, o)$ indicating the likelihood that the object $o = \{o_1, \dots, o_s\}$ is the nearest neighbor of $q = \{q_1, \dots, q_s\}$.

$$\begin{aligned}
 p_{nn}(q_1, o_1) &= (1-1/2) \cdot (1-2/2) = 0/4 \\
 p_{nn}(q_1, o_2) &= (1-2/2) \cdot (1-2/2) = 0/4 \\
 p_{nn}(q_2, o_1) &= (1-0/2) \cdot (1-0/2) = 4/4 \\
 p_{nn}(q_2, o_2) &= (1-0/2) \cdot (1-1/2) = 2/4 \\
 \hline
 p_{nn}(q, o) &= (4/4 + 2/4)/4 = 6/16 = 37,5\%
 \end{aligned}$$

Fig. 1. Computation of nearest-neighbor probabilities (s=2)

Lemma 3. Let $\{o_1, \dots, o_s\} \in C$. Then, the probability value $p_{nn}(q, o)$ indicating the likelihood that o is the nearest neighbor of q can be computed by:

$$p_{nn}(q, o) = \frac{\sum_{i,j \in 1 \dots s} p_{nn}(q_i, o_j)}{s^2},$$

where $p_{nn}(q_i, o_j)$ is equal to

$$\prod_{p \in C} \left(1 - \frac{\left| \{(q_i, p_l) \mid d(q_i, p_l) < d(q_i, o_j) \wedge l \in 1 \dots s\} \right|}{s} \right)$$

$p \neq q \wedge p \neq o$

Proof. First, we compute the probability $p_{nn}(q_i, o_j)$ that o_j is the closest sample to the sample q_i , by computing for each database object $p \in C$ the probability $P(p, q_i, o_j)$ that no sample of p is closer to the sample q_i than the sample o_j . Note that for objects $p \in DB \setminus C$ $P(p, q_i, o_j)$ is 1. The combination $\prod_{p \in C} P(p, o_i, o_j)$ of these independent probability values yields the probability that the sample point o_j is the nearest sample point for the sample point q_i . The average of these s^2 many probability values $p_{nn}(q_i, o_j)$ is equal to $p_{nn}(q, o)$. \square

In the following, show that the pruning distance for the uncertain query object q can further be decreased. The basic idea is that we do not use the minimal maximum object distance of q , i.e. d_{minmax} , but the minimal maximum distance of each single sample point.

Definition 4 (minimal maximum sample distance)

Let DB be a set of uncertain objects and let $q = \{\{q_{1,1}, \dots, q_{1,n_1}\}, \dots, \{q_{k,1}, \dots, q_{k,n_k}\}\}$ be a clustered query object representation. Then, the *minimal maximum sample distance* of each sample point $q_{i,j}$ and the *minimal maximum cluster distance* of each cluster $C_i = \{q_{i,1}, \dots, q_{i,n_i}\}$ are computed as follows:

$$\begin{aligned}
 d_{minmax}(q_{i,j}) &= \min \{ \text{maxdist}(q_{i,j}, \text{MBR}(o)) \mid o \in DB \} \\
 d_{minmax}(C_i) &= \min \{ \text{maxdist}(\text{MBR}(C_i), \text{MBR}(o)) \mid o' \in DB \}
 \end{aligned}$$

Lemma 4. Let DB be a set of uncertain objects. Then, the following statement holds for an uncertain query object $q = \{\{q_{1,1}, \dots, q_{1,n_1}\}, \dots, \{q_{k,1}, \dots, q_{k,n_k}\}\}$:

$$\forall i \in 1..k \ \forall j \in 1..n_i : d_{minmax}(q_{i,j}) \leq d_{minmax}(C_i) \leq d_{minmax}$$

In our final approach, we exploit the above lemma. Basically, our probabilistic nearest-neighbor query computes for the query object $q = \{\{q_{1,1}, \dots, q_{1,n_1}\}, \dots, \{q_{k,1}, \dots, q_{k,n_k}\}\}$ the possible nearest neighbors in the set DB by carrying out the following two steps for each $o \in DB$, an example is depicted in Figure 1:

ALGORITHM 1. Probabilistic-Nearest-Neighbor Query.

INPUT: $q = \{q_{1,1}, \dots, q_{1,n_1}\}, \dots, \{q_{k,1}, \dots, q_{k,n_k}\}\}$,
R-tree containing clustered uncertain objects from DB

OUTPUT: $(o, p_{nn}(q, o))$ for all objects $o \in DB$ where $p_{nn}(q, o) > 0$

BEGIN

- 1 **FOR ALL** $i \in 1..k$ **DO**
- 2 **FOR ALL** $j \in 1..n_i$ **DO**
- 3 LIST $nnlist(q_{i,j})$; // manages entries of the form $(o, p_{nn}(q_{i,j}, o), sample_cnt_o)$
- 4 PriorityQueue $queue$; // sorted in ascending order according to the mindist value of the entries
- 5 $queue.insert(mindist(MBR(q), MBR(R-tree.root)), (q, R-tree.root));$
- 6 **WHILE NOT** ($queue.isempty()$ **OR** ProbDoNotChange ($\{nnlist(q_{i,j}) | i \in 1..k \wedge j \in 1..n_i\}$)) **DO**
- 7 Element $first = queue.pop();$
- 8 **CASE** type($first.db$) // of what type is the R-tree element $first.db$?
- 9 DirNode, DataNode: // type of $first.query$ is Object
- 10 **FOR EACH** element IN $first.db$ **DO** // element is a tree node
- 11 $d = mindist(MBR(first.query), MBR(element));$
- 12 $queue.insert(d, (first.query, element));$
- 13 Object: // type of $first.query$ is also Object
- 14 **IF** SplitFurtherObject($first, queue$) **THEN**
- 15 **FOR EACH** $C_i(q)$ IN $first.query$ **DO**
- 16 **FOR EACH** $C_i(o)$ IN $first.db$ **DO** {
- 17 $d = mindist(MBR(C_i(q)), MBR(C_i(o)));$
- 18 $queue.insert(d, (C_i(q), C_i(o)));$
- 19 **ELSE** UpdateProbValues($first, \{nnlist(q_{i,j}) | i \in 1..k \wedge j \in 1..n_i\}$);
- 20 ObjectCluster: // type of $first.query$ is also ObjectCluster
- 21 **IF** SplitFurtherCluster($first, queue$) **THEN**
- 22 **FOR EACH** $q_{i,j}$ IN $first.query$ **DO**
- 23 **FOR EACH** $o_{i',j'}$ IN $first.db$ **DO**
- 24 $queue.insert(dist(q_{i,j}, o_{i',j'}), (q_{i,j}, o_{i',j'}));$
- 25 **ELSE** UpdateProbValues($first, \{nnlist(q_{i,j}) | C_i(q) = first.query \wedge j \in 1..n_i\}$);
- 26 ObjectSample: // type of $first.query$ is also ObjectSample
- 27 UpdateProbValues($first, \{nnlist(q_{i,j}) | q_{i,j} = first.query\}$);
- 28 **END;**
- 29 **END DO;**
- 30 ReportResults ($\{nnlist(q_{i,j}) | i \in 1..k \wedge j \in 1..n_i\}$);

END.

- First, we compute simultaneously for each sample point $q_{i,j}$ the probability $p_{nn}(q_{i,j}, o)$ that an object o is the nearest neighbor of the sample point $q_{i,j}$.
- Second, we combine the s probability values $p_{nn}(q_{i,j}, o)$ to an overall probability value $p_{nn}(q, o)$ which indicates the likelihood that the object o is the nearest neighbor of q .

The second task can be carried out straightforward based on the following lemma, whereas the first task is more complex and is explained in the remainder of this section.

Lemma 5. Let DB be a set of uncertain objects and let $q = \{q_{1,1}, \dots, q_{1,n_1}\}, \dots, \{q_{k,1}, \dots, q_{k,n_k}\}\}$ be an uncertain query object. Then, the following statement holds.

$$\forall o \in DB: p_{nn}(q, o) = \frac{1}{s} \cdot \sum_{i=1 \dots k, j=1 \dots n_i} p_{nn}(q_{i,j}, o)$$

Thus, the remaining question is how to compute the values $p_{nn}(q_{i,j}, o)$ efficiently. The approach proposed in this paper can be regarded as an extension of the nearest-neighbor search algorithm presented in [7]. Contrary to [7], our approach deals with complex clustered uncertain object representations instead of simple feature vectors. Furthermore, we do not compute a distance ranking for the query object q but a probability value $p_{nn}(q_{i,j}, o)$ to each sample point $q_{i,j}$ indicating the likelihood that object $o \in DB$ is nearest neighbor of $q_{i,j}$.

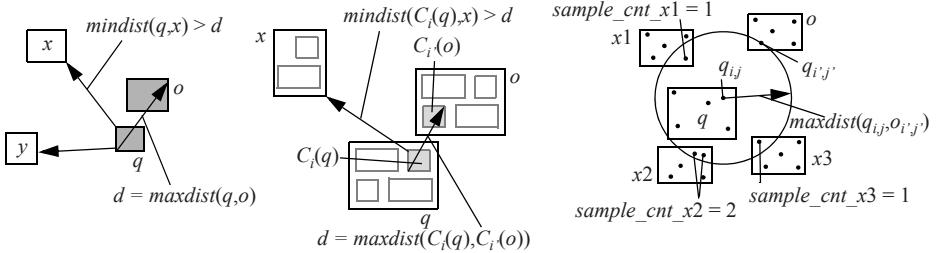
Algorithm 1 depicts our proposed *probabilistic nearest-neighbor query* algorithm. Like in the approach presented in [7], we use a priority queue *queue* as main data structure. In our case, the priority queue contains tuples $\text{entry} = (d, (q, o))$, where q is a part of the query object $\text{entry}.query$, o is a part of a database object $\text{entry}.db$, and d indicates the minimum distance between q and o . The distance values d determine the ordering of the priority queue. We have to store pairs of objects instead of simple objects because the query object itself consists of different parts, i.e. s sample objects $q_{i,j}$ and k clusters C_i (called $C_i(q)$ in the algorithm for clarity reasons). The priority queue is initialized with the pair $(\text{mindist}(\text{MBR}(q), \text{MBR}(\text{Rtree.root})), (q, \text{Rtree.root}))$. We always take the first element from the priority queue and test of what type the stored elements are. Then we decide for the first element of the priority queue whether it must be further refined or whether we can already use this first element to change the probability values of the probabilistic nearest neighbors of the query sample points $q_{i,j}$. Three cases are distinguished (cf. Figure 2):

- Assume the elements contained in the first element *first* of the priority queue are complete uncertain objects q and o . Then we test whether there exists an entry $(d, (p, p'))$ in *queue* for which the value d is smaller than $\text{maxdist}(q, o)$, using the function *SplitFurtherObject(first, queue)*. If this is the case, we split q and o into their cluster elements $C_i(q)$ and $C_i(o)$ and store the k^2 many combinations of these clusters in *queue*. If there does not exist such an entry $(d, (p, p'))$ (cf. Figure 2a), we update the lists $\text{nnlist}(q_{i,j})$ which contain all information about the up-to-now found probabilistic nearest neighbors of the sample point $q_{i,j}$. In the function *UpdateProbValues(first, {nnlist}(q_{i,j}) \mid i \in 1 \dots k \wedge j \in 1 \dots n_i)*, the entries $(o, p_{nn}(q_{i,j}, o), \text{sample_cnt_o})$ are updated. The values $p_{nn}(q_{i,j}, o)$ indicating the likelihood that o is the nearest neighbor of $q_{i,j}$ are set to (cf. Figure 2a):

$$\prod_{\substack{(x, p_{nn}(q_{i,j}, x), \text{sample_cnt_x}) \in \text{nnlist}(q_{i,j}) \\ x \neq o}} \left(1 - \frac{\text{sample_cnt_x}}{s}\right)$$

Furthermore, the values *sample_cnt_o* are set to s .

- Assume the elements contained in *first* are clusters, i.e. cluster $C_i(q)$ corresponds to the query object and cluster $C_i(o)$ corresponds to the database object. Then, in the function *SplitFurtherCluster(first, queue)*, we first test whether there exists an entry $(d, (p, p'))$ in *queue* for which the value d is smaller than $\text{maxdist}(C_i(q), C_i(o))$ and for which the following two conditions hold. First, p has to be equal to q , to $C_i(q)$, or to an object sample $q_{i,j}$. Second, p' must not be a part of o , i.e. another cluster of o or a sample point of o . If an entry $(d, (p, p'))$ exists for which these conditions hold, we split $C_i(q)$ and $C_i(o)$ in its sample points $q_{i,j}$ and $o_{i',j'}$ and store the $|C_i(q)| \cdot |C_i(o)|$ many combinations of the sample points in *queue*. If there does not exist such an entry



a) case 1: object pair (q, o) **b)** case 2: cluster pair $(C_i(q), C_i(o))$ **c)** case 3: update of the value $p_{nn}(q_{ij}, o)$
does not have to be refined does not have to be refined $p_{nn}(q_{ij}, o) = p_{nn}(q_{ij}, o) + 1/5 \cdot (4/5 \cdot 3/5 \cdot 4/5)$

Fig. 2. Three cases of the probabilistic nearest-neighbor query algorithm

$(d, (p, p'))$, we update the lists $nnlist(q_{ij})$ (cf Figure 2b). In the function *Update Prob-Values* (*first*, $\{nnlist(q_{ij}) \mid j \in 1 \dots n_i\}$), the entries $(o, p_{nn}(q_{ij}, o), sample_cnt_o)$ are updated. The values $p_{nn}(q_{ij}, o)$ indicating the likelihood that o is the nearest neighbor of q_{ij} are set to:

$$p_{nn}(q_{ij}, o) + \frac{|C_i(o)|}{s} \cdot \prod_{\substack{(x, p_{nn}(q_{ij}, x), sample_cnt_x) \in nnlist(q_{ij}) \\ x \neq o}} \left(1 - \frac{sample_cnt_x}{s}\right)$$

Furthermore, the values $sample_cnt_o$ are set to $sample_cnt_o \pm |C_i(o)|$.

- Assume the elements in *first* are sample points, i.e. q_{ij} is the query object and $o_{ij'}$ is the database object. Then, we call the function *UpdateProbValues* (*first*, $\{nnlist(q_{ij})\}$) which updates the entries $(o, p_{nn}(q_{ij}, o), sample_cnt_o)$. The values $p_{nn}(q_{ij}, o)$ indicating the likelihood that o is the nearest neighbor of q_{ij} are modified as follows (cf. Figure 2c):

$$p_{nn}(q_{ij}, o) + \frac{1}{s} \cdot \prod_{\substack{(x, p_{nn}(q_{ij}, x), sample_cnt_x) \in nnlist(q_{ij}) \\ x \neq o}} \left(1 - \frac{sample_cnt_x}{s}\right)$$

Furthermore, the values $sample_cnt_o$ are set to $sample_cnt_o + 1$.

The algorithm terminates, if either the priority queue is empty or if in all s lists $nnlist(q_{ij})$ there exists an entry $(o, p_{nn}(q_{ij}, o), sample_cnt_o)$ for which $sample_cnt_o = s$ holds. If this is the case, the probability values of all elements in the database do not change anymore. Thus, we can stop processing any further elements from *queue*. After the algorithm terminates, the values $p_{nn}(q_{ij}, o)$ contained in the lists $nnlist(q_{ij})$ indicate the probability that o is the nearest neighbor of q_{ij} . Finally, in accordance with Lemma 5, the probability values that o is the nearest neighbor of q are computed in the function *ReportResults* ($\{nnlist(q_{ij}) \mid i \in 1 \dots k \wedge j \in 1 \dots n_i\}$).

4 Experimental Evaluation

In this section, we examine the effectiveness, i.e. the quality, and the efficiency of our proposed probabilistic nearest-neighbor query approach. The efficiency of our approach

was measured by the average number of required distance computations per query object which dominate the overall runtime cost.

The following experiments are based on the same datasets as used in [9]. We used artificial datasets, each consisting of a set of 3- and 10-dimensional uncertain feature vectors. Additionally, we also applied our approaches to two distributed real-world datasets PLANE and PDB where the feature vectors were described by multi-dimensional boxes according to [8]. The following table summarizes the characteristics of the datasets:

Table 1. Characteristics of the datasets

dataset	ART3 (low)	ART3 (high)	ART10 (low)	ART10 (high)	PLANE	PDB
dimensions d	3	3	10	10	42	120
uncertainty u	3%	5%	3%	4%	1%	4%

For the sampling of the possible object positions we assumed an equal distribution within the corresponding uncertainty areas. All d -dimensional datasets are normalized w.r.t. the unit space $[0,1]^d$. As distance measure we used the L_1 -distance (Manhattan distance). We split all datasets into two sets containing 90% respectively 10% of all objects. For the nearest neighbor queries, we used the objects from the smaller set as query objects and summarized the results. If not stated otherwise, the size of the sample set of each uncertain object is initially set to 25 samples which are approximated by 7 clusters.

4.1 Experiments on the Sample Rate

First, we turned our attention to the quality of our probabilistic nn-query approach by varying the number of used samples per object. We noticed that for sample rates higher than 100 the resulting probability values do not change any more considerably. Therefore, we used the probabilistic nn-query result $R_{exact} = \{(o, P_{exact}(q \theta_d o)) | P_{exact}(q \theta_d o) > 0\}$ (cf. Definition 1) based on 100 samples as reference query result for measuring the error of the probabilistic nn-query results $R_{approx} = \{(o, P_{approx}(q \theta_d o)) | P_{approx}(q \theta_d o) > 0\}$ based on sample rates $s < 100$. The used error measure Err_{nn} for the nearest-neighbor query is defined as follows:

$$Err_{nn}(R_{approx}, R_{exact}) = \sum_{q \in Q} \left(\frac{1}{2} \cdot \sum_{(q, o) \in Q \times DB} |P_{approx}(q \theta_d^{nn} o) - P_{exact}(q \theta_d^{nn} o)| \right)$$

Figure 3a shows the error of the probabilistic nearest-neighbor query for a varying sample rate s . It can be clearly seen that the error decreases rapidly with increasing sample rate s . At a sample rate $s = 10$ the error is less than half the size compared to the error at $s = 1$ for some datasets. Furthermore, comparing the artificial datasets with high uncertainties (ARTd(high)) to those with low uncertainties (ARTd(low)), we can observe that a higher uncertainty leads to a higher error.

In the next experiment, we investigated how the sample rate influences the cost of the query processing. Figure 3b shows the number of distance computations required to

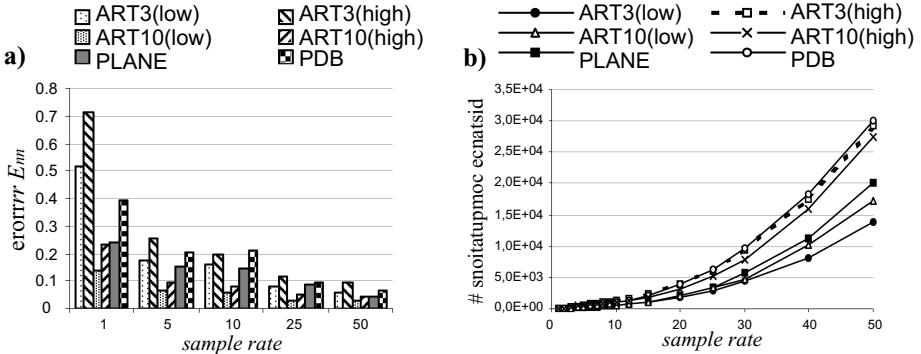


Fig. 3. Influence of the sample rate. a) error Err_{nn} , b) number of distance computations.

perform the nn-query for varying sample rates. We set the number k of clusters to 5 for a sample rate s higher than 5, otherwise we set $k = s$. The cost increase superlinear with increasing sample rates s . For high sample rates, the good quality (cf. Figure 3a) goes along with high query cost (cf. Figure 3b). In particular, the query processing on datasets with high uncertainty ($ARTd(high)$) does not only lead to a lower quality of the results but is also more expensive than the processing on more accurate datasets ($ARTd(low)$). In the case of very uncertain datasets the computational cost are higher because the pruning distances, i.e. the *minimal maximum object distances* (cf. Definition 3), for very uncertain objects are much higher than for non-uncertain objects. Altogether, we achieve a good trade-off between the quality of the results and the required cost when using a sample rate of $s = 25$.

4.2 Experiments on the Efficiency

Next, we examine the runtime performance of our probabilistic nearest-neighbor query approach. Figure 4 shows how the runtime performance depends on the number k of sample clusters. On the one hand, when using only one cluster per object ($k = 1$), we have only a few clusters for which we must compute the distances between them. This is due to the fact that the cluster covers the entire uncertain object, i.e. it has a large extension.

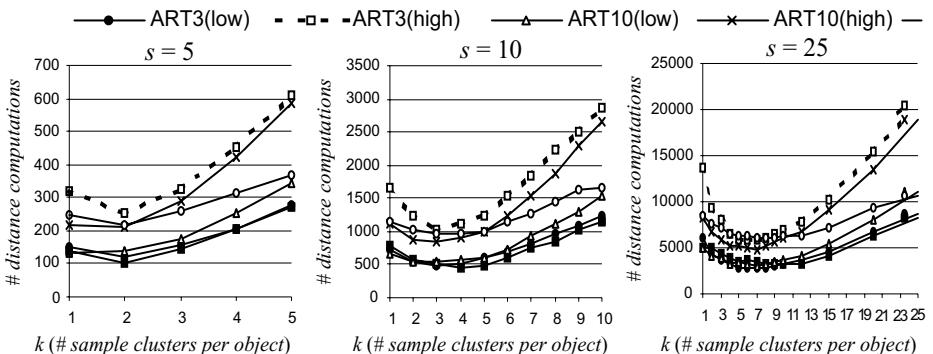


Fig. 4. Runtime performance for varying number of sample clusters

On the other hand, very small clusters ($k = s$) also lead to an expensive query processing, because we have to compute a lot of distances between pairs of clusters when refining the object pairs. The best trade-off for k can be achieved somewhere in between these two extremes. As depicted in Figure 4, the optimal setting for k depends on the used sample rate. Generally, the higher the used sample rate s , the higher is the optimal value for k . Note that the \maxdist values of the cluster pairs are very high when using $k = 1$ sample clusters. In this case, we often have to investigate the corresponding sample points of the clusters which leads to a high number of distance computations. Table 2 shows the ratio between the cost required for $k = 7$ and $k = 1$ for the probabilistic nearest-neighbor query (θ_d^{mn}) ($s = 25$). We can conclude that the clustering of the object samples pays off when using an adequate choice of the parameter k .

Table 2. Cost ratio between $k = 7$ and $k = 1$ ($s = 25$)

datasets	ART3 (low)	ART3 (high)	ART10 (low)	ART10 (high)	PLANE	PDB
θ_d^{mn}	0.46	0.43	0.61	0.60	0.64	0.71

In the last experiment, we compare our efficient probabilistic nearest-neighbor query approach (*accelerated approach*) as presented in Algorithm 1 to the straightforward solution (*simple approach*) which takes for the pruning of candidate pairs solely the minimal maximum object distance into account (cf. Definition 3 and Lemma 3). Figure 5 depicts the results for the query processing on different datasets and varying sample rates. Figure 5a shows that we achieve a very significant reduction of the query cost using the pruning techniques of our probabilistic nearest-neighbor query algorithm independent of the used sample rate. For the ART10(*high*) dataset the cost were reduced to 20% and for both real-world datasets we even achieved a reduction to 15%. Figure 5b compares the performance of both approaches using the artificial datasets for varying uncertainties of the objects. This experiment shows that the simple approach is not applicable for high uncertainties due to the enormous number of required distance computations. Contrary, our accelerated approach is not very sensitive to the uncertainty of the objects and shows good performance even for very imprecise data.

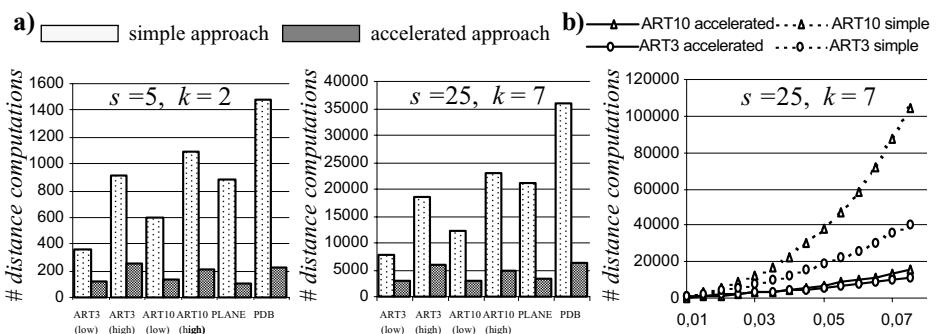


Fig. 5. Runtime performance for different pruning techniques

5 Conclusions

Probabilistic query processing on uncertain data is an important emerging topic in many modern database application areas. In this paper, we introduced an approach for computing probabilistic nearest-neighbor queries on uncertain objects which assigns to each object a probability value indicating the likelihood that it belongs to the result set. We showed how this probabilistic query can effectively be carried out based on the generally applicable concept of monte-carlo sampling, i.e. each uncertain object is described by a set of sample points. In order to improve the query performance, we determined appropriate approximations of the object samples by means of clustering. Minimal bounding boxes of these clusters, which can be efficiently managed by spatial index structures, are then used to identify and skip unnecessary distance computations in a filter step. In a detailed experimental evaluation based on artificial and real-world data sets, we showed that our technique yields a performance gain of a factor of up to 6 over a straightforward comparison partner.

In our future work, we plan to extend our probabilistic algorithms to join processing, which built a foundation for various data mining algorithms, e.g. clustering and classification on uncertain data.

References

- [1] Abiteboul S., Hull R., Vianu V.: *Foundations of Databases*. Addison Wesley, 1995.
- [2] Böhm, C., Pryakhin A., Schubert M.: *The Gaus-Tree: Efficient Object Identification of Probabilistic Feature Vectors*. ICDE'06.
- [3] Cheng R., Kalashnikov D.V., Prabhakar S.: *Evaluating probabilistic queries over imprecise data*. SIGMOD'03.
- [4] Cheng R., Kalashnikov D. V., Prabhakar S.: *Querying imprecise data in moving object environments*. IEEE Transactions on Knowledge and Data Engineering, 2004.
- [5] Dai X., Yiu M., Mamoulis N., Tao Y., Vaitis M.: *Probabilistic Spatial Queries on Existentially Uncertain Data*. SSTD'05.
- [6] Guttman A.: *R-trees: A Dynamic Index Structure for Spatial Searching*. SIGMOD'84.
- [7] Hjaltason G. R., Samet H.: *Ranking in Spatial Databases*. SSD'95.
- [8] Kriegel H.-P., Kunath P., Pfeifle M., Renz M.: *Approximated Clustering of Distributed High Dimensional Data*. PAKDD'05.
- [9] Kriegel H.-P., Kunath P., Pfeifle M., Renz M.: *Probabilistic Similarity Join on Uncertain Data*. DASFAA'06.
- [10] McQueen J.: *Some Methods for Classification and Analysis of Multivariate Observations*. In 5th Berkeley Symp. Math. Statist. Prob., volume 1, 1967.
- [11] Motro A.: *Management of Uncertainty in Database Systems*. In Modern Database Systems, Won Kim (Ed.), Addison Wesley, 1995.
- [12] Wolfson O., Sistla A. P. , Chamberlain S., Yesha Y.: *Updating and Querying Databases that Track Mobile Units*. Distributed and Parallel Databases, 7(3), 1999.
- [13] Zhao W., Chellappa R., Phillips P.J., Rosenfeld A.: *Face Recognition: A literature survey*. ACM Computational Survey, 35(4), 2000.

Making the Most of Cache Groups

Andreas Bühmann and Theo Härdter

Department of Computer Science, University of Kaiserslautern,
P. O. Box 3049, D-67653 Kaiserslautern, Germany
`{buehmann, haerder}@informatik.uni-kl.de`

Abstract. Cache groups are a powerful concept for database caching, which is used to relieve the backend database load and to keep referenced data close to the application programs at the “edge of the Web”. Such cache groups consist of cache tables containing a subset of the backend database’s data, guided by cache constraints. If certain query types are anticipated in the application workload, specifically designed cache groups can directly process parts of incoming declarative queries. The main class of such queries, project-select-join queries, can be supported by specifying a proper set of referential cache constraints.

Cache groups should be managed in the most cost-effective way. Hence, redundant constraints should not be respected during cache loading and consistency maintenance to avoid unnecessary overhead. On the other hand, because as much queries as possible should be processable in the cache, all redundant relationships implied by the set of specified cache constraints should be made explicit to help the query optimizer.

1 Database Caching with Cache Groups

Often, performance of data-intensive applications over wide-area networks (e.g., of transactional Web applications or Web information systems) is limited by the (backend) database (DB), especially by its processing power, its resource availability, and the communication delays for serving user requests. A proven remedy for such situations is the use of caching to substantially increase scalability and availability of the system as well as to drastically reduce the user-perceived delays of information requests.

Web caching, as another kind of caching in this context,  typically keeps static Web objects (XML fragments or images) in some of the caches in the user-to-server path and only enables identifier-based requests for cached objects. In contrast, DB caching is intended to deliver correct results for declarative DB queries (e.g., in SQL) from the current cache contents, thereby relieving the backend DB of some of its workload. Latency of user requests is supposed to be noticeably reduced by allocating these caches close to the application servers at the edge of the Web. This, however, only happens if user queries can be completely evaluated in the cache to save the travel times of messages (query shipping, result transmission) to the backend DB through wide-area networks. Hence, analysis of workloads must help to determine the future data reference behavior of applications and, in this way, prepare for appropriate locality of reference for them, to enable cache-based answering of frequent queries. This task is often facilitated by geographic contexts, which frequently determine the workload of application

servers and, in turn, the data to be kept in their DB caches. Because these caches need powerful functionality for query optimization and processing, storage management, indexing, etc., they are often managed by full-fledged DBMSs and are therefore called frontend DBs, too.

Differing from approaches that make use of (stacked) materialized views [2], our cached data is organized in cache groups [3], which consist of a set of cache tables. These cache tables contain a subset of the backend DB's data, whose selection is guided by cache constraints – the approach therefore also being called *constraint-based DB caching*. It primarily rests upon referential cache constraints (RCCs), which specify the data sets needed to run selected project-select-join (PSJ) queries in the cache. Such specifications may be redundant or may contain RCC cycles, which imply cache groups exhibiting non-minimal maintenance or excessive loading or unloading [4].

Our specific contribution in this paper is to introduce a set of rules for proper cache group design and usage. In case of redundantly specified RCCs, our rules identify these redundant constraints, which will reduce cache maintenance overhead. On the other hand, our rules derive all redundant relationships implied by the set of specified cache constraints and make them explicit. This facilitates the query optimizer's task of figuring out all (parts of) queries that can be evaluated in the cache – besides those ones the cache groups are designed for.

The rest of the paper is organized as follows: In the following Sect. 2 we illustrate how cache groups are designed, how they are loaded, and how they are probed in order to determine whether a given query can be processed in the cache. Section 3 derives the set of rules that govern the optimization of cache groups, whereas we apply these rules to a sizeable example in Sect. 4 to demonstrate the course and the effects of this optimization process. Finally in Sect. 5 we summarize our results and give an outlook on our future work.

2 Designing Cache Groups

The key idea of constraint-based database caching is to accomplish *predicate completeness* for some given types of query predicates P in the cache such that all queries matching P can be evaluated correctly. This technique does not rely on static predicates: Parameterized constraints make the specification adaptive; so-called candidate values (CVs) are used to instantiate the corresponding parameters: An “instantiated constraint” then corresponds to a predicate and, once the constraint is satisfied (i.e., all related records have been loaded), it delivers correct answers to eligible queries. Hence, the candidate values should be carefully chosen, because they determine the set of cache-evaluable predicates. They describe the future reference locality anticipated in the cache and, therefore, serve as a kind of “loading directives” for the cache manager.

2.1 Basics of Cache Groups

A cache contains a collection of cache tables, which represent backend tables and which can either be isolated or related to each other in some way. For simplicity, let the table and column names be the same in the cache and in the backend DB: Considering a cache table S , S_B designates its corresponding backend table, $S.c$ a column c of S . All

records (of various types) in the backend DB that are needed to evaluate predicate P are called the *predicate extension* of P .

For comprehension, let us repeat some definitions from [4]: The simplest form of predicate completeness is *value completeness*. A value v is said to be value complete (or *complete* for short) in a column $S.c$ if and only if all records of $\sigma_{c=v} S_B$ are in S . Hence, if we know that a value v is value complete in a column $S.c$, we can correctly evaluate $S.c = v$, because all records from table S_B that carry this value are in the cache. Furthermore, if we know that all values occurring in a column $S.c$ are complete, we call $S.c$ *column complete*. This property allows to evaluate all simple equality predicates $S.c = x$ in the cache as soon as a value x is found in $S.c$.

To answer PSJ queries in the cache, we must be sure that their extensions are present. Specific *equi-join predicates* can be evaluated only if all corresponding join partners are in the cache, which is enforced by using *referential cache constraints* (RCCs) [3]. An RCC is defined between two cache columns not necessarily belonging to separate tables. An RCC $S.a \rightarrow T.b$ from a source column $S.a$ to a target column $T.b$ is satisfied if and only if all values v in $S.a$ are value complete in $T.b$.

This RCC ensures that, whenever we find a record s in cache table S , all join partners of s with respect to $S.a = T.b$ are in T , too. Note, the RCC alone does not allow us to perform this join in the cache correctly: Many records of S_B that have join partners in T_B may be missing from S . But using an equality predicate with a complete value in column $S.c$ as an anchor, we can restrict this join to pairs of records that are present in the cache: The RCC $S.a \rightarrow T.b$ expands the predicate extension of $S.c = x$ to the predicate extension of $S.c = x \wedge S.a = T.b$. In this way, a complete value can serve as an *entry point* into the cache for the evaluation of a query; it allows us to start reasoning about predicates evaluable in the cache: Once the cache has been entered in this sense, reachable RCCs show us where joins can correctly be performed: Of course, the application of RCCs can be chained.

A column is non-unique (NU) by default, but it can be declared unique (U) via the SQL constraint *unique* in the backend DB schema. Depending on the types of source and target columns, RCCs of types $1:n$, $n:1$, and $n:m$ may occur.

Probing is the process of finding out whether, given an equality predicate $S.c = v$ in a query, the value v is complete in column $S.c$. This knowledge is the foundation for applying RCCs along the join directions that occur in the query. There are basically two approaches to probing that can be combined to form probing strategies:

- If $S.c$ is known to be column complete, it suffices to check whether v exists in $S.c$. If it exists, it is complete.
- Otherwise RCCs can be exploited: If v exists in one of the source columns of RCCs leading to $S.c$, the value v is complete (in $S.c$).

2.2 Loading the Cache

How do we fill the cache? To initiate cache loading, we have to specify some filling columns $S.f$: Assume $x \in S_B.f$ is in the CV list and the cache manager wants to instantiate a cache constraint containing $S.f = x$. In a first step, x is made complete, which loads a number of records into S . Then for each RCC $S.a \rightarrow T.b$ emanating from S , the

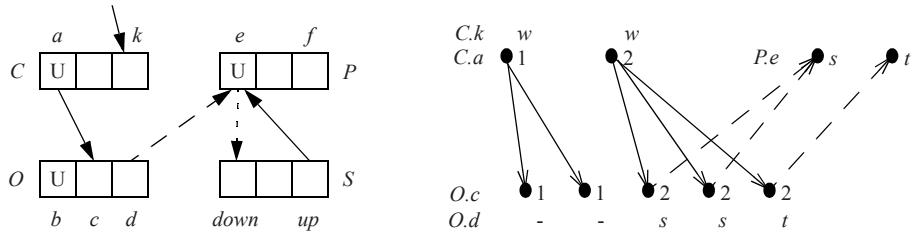


Fig. 1. Cache group *COPS*: Construction of a predicate extension for *COP*

newly inserted values in *S.a* have to be made complete in *T.b*. Hence, new records are inserted into all target tables *T_i* reached by RCCs originating from *S*. In the same way, RCCs emanating from *T_i* provoke loading actions in further cache tables, until all RCC constraints are satisfied.

We can use cache tables, filling columns and RCCs to specify *cache groups*, which is our unit of design to support a specific predicate type in the cache. A cache group is a collection of cache tables linked by a set of RCCs. A distinguished cache table is called the *root table R* and holds one or more filling columns. The remaining cache tables are called *member tables* and must be reachable from *R* via RCCs.

With these definitions, we are able to introduce predicate extensions for PSJ queries. First let us discuss the loading process in detail by an example: Cache group *COPS* (Customer, Order, Product, Structure) in Fig. 1(left), which includes two RCCs of type $1:n$ and two RCCs of type $n:1$ (arrows). For a moment forget table *S* and both RCCs between *S* and *P*. Then assume the predicate of a PSJ query to be evaluated on *COP* is

$$Q = (C.k = w \wedge C.a = O.c \wedge O.d = P.e).$$

An example of *Q*'s predicate extension is sketched in Fig. 1(right), where dots represent records, lines value-based relationships. To establish value completeness for the value *w* of filling column *C.k*, the cache manager loads all records of $\sigma_{k=w} C_B$ in a first step. For each of these records loaded, the RCC *C.a* \rightarrow *O.c* must be fulfilled (primary key/foreign key relationships, solid lines); that is, all values of source column *C.a* (1, 2 in the example) must be made complete in the target column *O.c*. Finally, for all values present in *O.d* (*s*, *t*), the RCC *O.d* \rightarrow *P.e* makes their counterparts complete in *P.e* (foreign key/primary key relationships, dashed lines). Hence, we have constructed the predicate extension needed for *Q* exactly.

To make cache group design more elegant, we simplify our specification concepts: Those values of the CV list that have already initiated cache loading may be considered as values in artificial control columns and their relationships to filling columns may be described by RCCs. (For example, the RCC stub leading from nowhere to *C.k* in Fig. 1 indicates such an RCC; we leave out the artificial columns in our figures.) With this unification of cache group specification, cache tables are loaded only via RCCs. Following the RCCs, the cache manager can construct predicate extensions using only simple loading steps based on equality of values. Accordingly, it can correctly evaluate the corresponding queries locally.

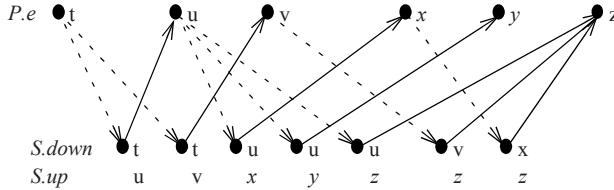


Fig. 2. Unsafe loading of products in *COPS*. Dots represent records, lines value-based relationships along RCCs (line patterns indicate the responsible RCC from Fig. 1).

We will show by an example that, for reasons of “safe” cache loading and maintenance, not all cache groups are acceptable: Assume we continue to load *COPS*, where tables P and (now) S contain the bill-of-material representation of products. As soon as value t is made complete in P (via RCC $O.d \rightarrow P.e$), it initiates loading in S via $P.e \rightarrow S.down$ to make t complete in $S.down$. In turn, this action loads values u and v into $S.up$, which enforces completeness for these values in $P.e$ via $S.up \rightarrow P.e$. As illustrated in Fig. 2, cache loading recursively iterates over the RCC cycle and causes product t and its entire composed-of hierarchy to be loaded into the cache. Such excessive load situations are called *unsafe* and are prohibited when designing cache groups [4].

An RCC cycle is classified as *homogeneous* or *heterogeneous*, if it involves only a single column or more than one column in some participating table, respectively. If several cycles occur in a cache group and influence each other, some records loaded via a cycle may smuggle values into other cycles, which may keep these cycles running. Therefore, as proven in [4], while isolated homogeneous RCC cycles are acceptable, other cyclic RCC specifications must be prohibited to prevent unsafe cache groups:

- Isolated heterogeneous RCC cycles are not allowed.
- Heterogeneous RCC cycles with non-compensating smuggler relationships are not allowed.

The RCC cycle in cache group *COPS* (Fig. 1) is heterogeneous and isolated and should hence not be part of a cache group design.

3 Optimizing the Design

Making the most of a given cache group has two facets: First, when answering queries, we would like the query evaluation in the cache to be as powerful and flexible as possible. Second, when maintaining the cache contents – in order to fulfill the defined cache constraints – or when probing, we try to get by with the least possible effort.

3.1 Utilizing Redundancy

The path to both of these optimization goals lies in discovering redundancy in the cache group: Excluding redundant paths of loading steps during maintenance avoids unnecessary costs; including all possible (redundant) join directions enables the query analysis

to use the cached predicate extensions for a greater variety of queries. Therefore, we need to know where redundant RCCs are.

Additionally, knowledge about column-complete columns as well as about redundant RCCs offers more and probably cheaper possibilities for probing [4]: Redundant RCCs need not be used during probing, and using a column-complete column, one is able to avoid considering RCCs altogether.

An RCC is called redundant if dropping it from the cache group does not change the cache group's behavior with regard to record loading: The same sets of records will be present in the cache in any situation after any number of loaded CVs, with or without the redundant RCC. Every RCC is either a *redundant RCC* (RRCC) or a *non-redundant RCC* (NRCC).

3.2 Optimization Rules

In summary, given a cache group, we would like to find out

- which redundant RCCs can be added,
- which user-defined RCCs are in fact redundant and which are not,
- and which columns are complete.

Our goal is, on the one hand, to find an irreducible core of cache constraints that minimizes maintenance costs. On the other hand, we try to extend this core with a maximum of information that is useful during non-maintenance tasks.

To this end, we transform the user-specified cache group by applying a number of rules. These rules match certain situations in a cache group and may mark a column as column complete or introduce redundant RCCs. It is important that no rule ever changes the behavior of the cache group.

At the beginning of this optimization, due to the lack of better knowledge, we consider all user-defined RCCs non-redundant. When a newly discovered RRCC coincides with a user-defined NRCC, it effectively degrades the NRCC to an RRCC.

Figure 3 illustrates the situations in which our rules apply. The depicted tables and columns match the ones used in the textual descriptions of the rules below. We will walk through them one by one.

Unique Columns. We have two rules to discover complete columns. The first one is trivial, but it is needed nonetheless, because finding all complete columns is a prerequisite for successful application of some of the subsequent rules.

Rule 1. Every unique column is column complete. (Fig. 3a)

Every value in a unique column is complete as soon as it appears in the cache. Obviously, the column must always be complete then.

Induced Column Completeness. Our second rule deals with complete columns that are *induced* by RCCs and the loading mechanism.

Rule 2. Let $T.b$ be the only column of a table T that is reached by incoming NRCCs. Then $T.b$ is column complete. (Fig. 3b)

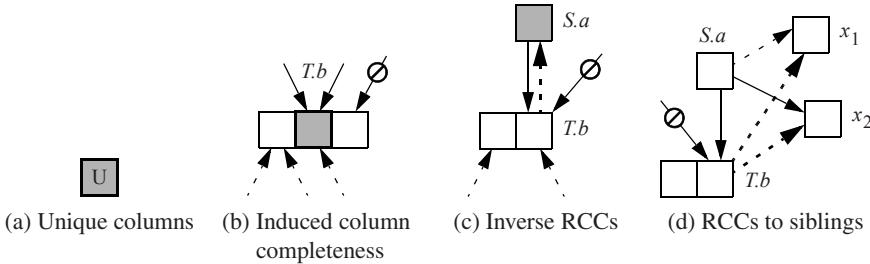


Fig. 3. Rules. Changes in the cache group are highlighted with thicker lines. The prohibition sign (\emptyset) marks exemplary RCCs that are not allowed for the rule to apply. (Complete columns are gray, redundant RCCs dotted.).

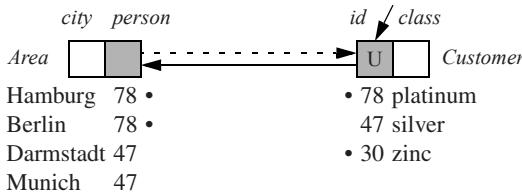


Fig. 4. Induced column completeness (of column $Area.person$) and inverse RCC ($Area.person \rightarrow Customer.id$). Records marked with a dot (•) are in the cache.

Every value that is loaded into T through one of the incoming NRCCs is complete in $T.b$. Since records are not loaded into T in another way (possibly existing RRCCs do not contribute to the loading), $T.b$ is column complete.

Let us look at a little example: Figure 4 shows a cache group comprising two cache tables $Customer$ and $Area$ (with their backend and cache contents) as well as an RCC $Customer.id \rightarrow Area.person$. Table $Customer$ is filled via column id . Customers 78 and 30 have been inserted, for each of which the corresponding $Area$ records have been loaded: two records for 78 (making 78 complete), none for 30 (assuming 30 is not in $Area_B$) which is therefore complete in $person$ nevertheless. Therefore, $person$ is column complete.

Column $person$ would stay complete if another incoming RCC were added to it (and made 47 complete, for example). But if (Munich, 47) were loaded because of an RCC to $city$, it could not be guaranteed that the other 47 record would get into the cache, too. Hence, 47 would not be complete and neither would $person$. Note that any number of incoming RRCCs are acceptable; RRCCs do not contribute to the loading of cache tables and, thus, are unable to challenge column completeness.

Inverse RCCs. An RCC $x \rightarrow y$ expresses that every value in x is complete in y . We can discover additional RCCs if we are able to control the set of values present in x (and can then show that these values have to be complete in y). The simplest situation where it is clear which values appear in x is when x 's table is loaded only via a single RCC $s \rightarrow x$ pointing to x . Then the values in x depend directly on the values in s : More precisely, x

can only contain a subset of values of s . Therefore, we say that a column $T.b$ is *column dependent* on a column $S.a$ iff the only NRCC targeting table T is $S.a \rightarrow T.b$.

By comparing this definition with Rule 2, it is obvious that every column-dependent column is complete, but not every complete column is column dependent.

Let us return to our example in Fig. 4. There we have two column-dependent columns, *person* and *id*. We will concentrate on *person*: Due to the incoming RCC, it contains a subset (78) of the values in *id* (30,78), which we know to be complete in *id*, because *id* has a unique constraint. Therefore, every value in *person* is complete in *id* and we can add an *inverse RRCC* $\text{person} \rightarrow \text{id}$.

The column *person* must not be reached by another NRCC (as opposed to our previous case of only column completeness), because a so-loaded 47 in *person* would not necessarily become complete in *id*.

We wrap up the sketched situation in our next rule:

Rule 3. Let $T.b$ be column dependent on a column $S.a$ due to an NRCC $S.a \rightarrow T.b$. If $S.a$ is column complete, then an inverse RRCC $T.b \rightarrow S.a$ holds. (Fig. 3c)

RCCs to Siblings. In special situations, two or more columns are in some sense synchronized due to RCCs originating from a common column. In Fig. 3d this common column is $S.a$ and we have got three RCCs leading from it to some other (child) columns $T.b$, x_1 , and x_2 . (The RCC $S.a \rightarrow x_1$ is redundant, the other two are not.) This means that all the values in $S.a$ are complete in all of these three columns; let $V_{S.a}$ denote this set of values.

As we know from the discussion of Rule 3, column dependency of a column, say $T.b$, restricts the set of values in this column to a subset (of $V_{S.a}$). Hence, every value in column $T.b$ is complete in the children of $S.a$, which we can express by redundant RCCs from $T.b$ to its siblings. (Strictly speaking, we could also add a redundant RCC from $T.b$ to itself. But because such an RCC can be equivalently replaced with a column-completeness label, we omit it: This would just be a special case of Rule 2.)

These thoughts leave us with the following rule:

Rule 4. Let $T.b$ be column dependent on a column $S.a$ due to an NRCC $S.a \rightarrow T.b$. Then for every column c_i that is reached by an RCC $S.a \rightarrow c_i$ from the same source column (i.e., a sibling of $T.b$), an additional RRCC $T.b \rightarrow c_i$ holds. (Fig. 3d)

Possible Extensions. Our rules do not find every redundant RCC possible. We will discuss two conceivable generalizations of existing rules that would enable us to find more redundant RCCs.

The example shown in Fig. 5a generalizes the situation that is covered by our Rules 3 and 4: Column $T.b$ is reached by *two* different homogeneous paths (where there is no change of column in any table on the path), both emanating from $S.a$.

This means that $T.b$ is not column dependent on $S.a$ according to our simple definition, but in a more general sense it is: The values in $T.b$ are still determined only by the values in $S.a$; on the paths towards $S.a$, more values may get lost than in our simple single-RCC case, but we still have a subset relationship. This means that an inverse RCC $T.b \rightarrow S.a$ is possible as well as RCCs from $T.b$ to the direct children of $S.a$ (which are no longer siblings of $T.b$).

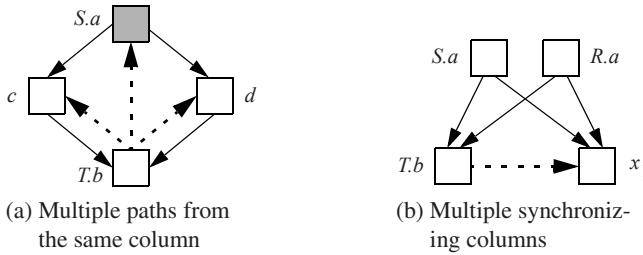


Fig. 5. Harder optimization RCCs

Figure 5b shows a different kind of generalization of the concept of dependency: This time, the values in column $T.b$ depend on both the values in $S.a$ and the values in $R.a$ (i.e., at any time, $T.b$ contains a subset of the union of those values). This setting still permits RCCs to siblings to be added, as long as these siblings (e.g., x_1) are reached by RCCs from each of these synchronizing columns.

Rules expressing the sketched situations are not as easily checked as our chosen ones, which can consider a column and its immediate neighborhood locally. In contrast, here we would have to collect information about paths of any length and compare sets of influencing columns. It is questionable whether these special situations occur often enough to justify the added complexity of the rules for their optimization.

3.3 Applying the Rules

How do we apply our four rules to a given cache group? The basic idea is simple: Keep applying the set of rules until no further match occurs and the cache group is in a stable and, with regard to our rules, optimized state. Obviously, we must be sure that this will happen eventually: Our rule application algorithm should not run into endless cycles.

Let us analyze the dependencies among our rules: Rules 3 and 4 produce RRCCs, which may override NRCCs. NRCCs eventually embody the irreducible core of the constraints; they are not produced by any rule. Since RRCCs are not removed, their number is only increasing, the number of NRCCs decreasing. This may at most lead to further columns becoming column dependent, which might make Rules 3 and 4 applicable again. This process is bounded by the number of feasible RCCs.

The first two rules only produce column-complete columns: Only Rule 3 depends on these column-complete columns. Since no rule removes the column-completeness status of a column, no cyclic behavior is possible – as long as we are careful enough to check whether a rule application did actually change the cache group.

Unique columns cannot be created during optimization: Therefore, Rule 1 can be independently applied in advance, before the other rules are applied repeatedly until there are no further changes to the cache group.

In a Java implementation [5], we have chosen to apply our rules according to a depth-first search of the cache group, starting at the filling columns and stopping when cycles are detected. This is sufficient, because all tables not reachable this way will not be filled and used either. Furthermore, we are able to analyze the cycles encountered during rule

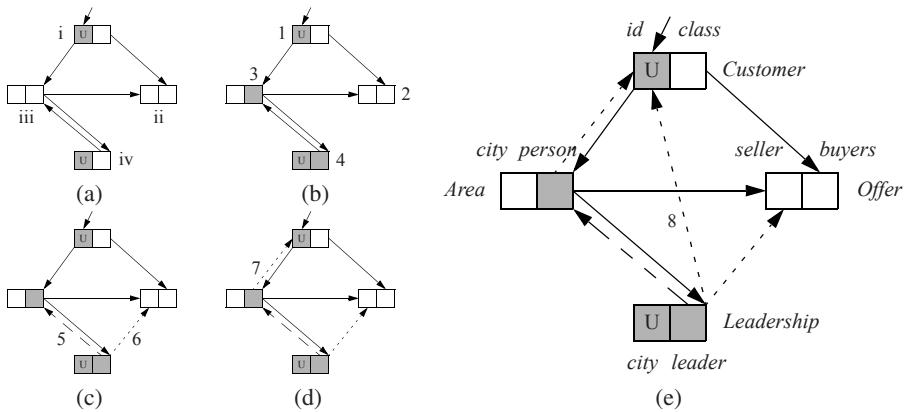


Fig. 6. Optimizing a cache group. The dashed lines indicate user-defined but redundant RCCs.

application and see whether they lead to controllable loading behavior or make for unsafe cache groups.

4 Example

Let us see our rules acting in concert to optimize a given cache group. Figure 6 depicts our object of optimization (a) as well the optimization result (e); we will show step by step how this result has been derived.

We start with a cache group that has been specified by someone who wants to use our caching system for his online selling platform (Fig. 6a): We have four cache tables, two unique columns *Customer.id* and *Leadership.city*, one filling column *Customer.id* and six user-defined RCCs, which we have to consider non-redundant until further investigation. (In Fig. 6e these are the five solid RCCs and the dashed one.)

In a preparing step, we apply Rule 1 to every column in the cache group. The order in which we visit the cache tables in this and all the following steps is given in Fig. 6a in Roman figures: a depth-first search starting at the filling column *C.i*. (In the following, we abbreviate table and column names by their first letters.) In this way, we find the unique columns *C.i* and *L.c* and mark them as column complete.

We then begin to apply Rules 2-4 to the cache group (Fig. 6b):

1. *C.i* is the only column of *C* that is reached by NRCCs; therefore, by Rule 2, it is column complete (which we already know, so this does not change the cache group). *C.i* is column dependent as well, but because it is dependent on an artificial column outside of our main cache group, we skip the other rules.
2. In table *O* no column is induced column complete or column dependent, because there are incoming NRCCs on two columns: None of our rules matches.
3. *A.p* is reached by two NRCCs, but not any other column is: *A.p* is column complete. Note that *A.p* is not column dependent, because it is influenced by both *C.i* and *L.l*.
4. *L.l* is induced column complete because of the only NRCC *A.p* → *L.l*, which makes *L.l* column dependent, too.

5. Since $L.l$ is column dependent, we can add an inverse RRCC $L.l \rightarrow A.p$ (Rule 3). This degrades the already existing NRCC (indicated by a dashed line in Fig. 6c), recognizing it as redundant.
6. According to Rule 4 we can finally add redundant RCCs from $L.l$ to each of its siblings (with respect to the common father column $A.p$): Here this makes only for one RRCC $L.l \rightarrow O.s$.

This concludes our first run through the cache group; we have visited each table once. Since we have made four changes (two column-complete columns and two RRCCs), and because these might have established the preconditions for further rule applications, we have to start a second time: In tables C and O we come across the same states as before, but in A we find something new:

7. Column $A.p$ has become column dependent on $C.i$ due to the degradation of the former NRCC $L.l \rightarrow A.p$. This means – according to Rule 2 – that we can add an inverse RRCC $A.p \rightarrow C.i$ (Fig. 6d). Furthermore, we could add RRCCs to children of $C.i$ if there were any besides $A.p$.
8. In table L column $L.l$ is still column dependent on $A.p$ – as discovered in step 4 (A column can never lose its column dependency.) In step 6 we have already applied Rule 4 and introduced RRCCs to all siblings of $A.p$ – but wait, there is a new sibling, namely $C.i$, due to the recently created RRCC $A.p \rightarrow C.i$. Hence, we can add an RRCC $L.l \rightarrow C.i$ back to the *Customer* table (Fig. 6e).

Our second run through the cache group is finished. We have added two RRCCs and must therefore perform a third run to see whether these changes have opened up further possibilities. You should be able to verify that this is not the case. Accordingly, the state in Fig. 6e is our optimized version of the cache group the user has defined:

- We have identified three additional RCCs, which, during query analysis and evaluation, allow for more join directions in the cache. For example, the predicate $L.c = \text{'Berlin'} \wedge L.l = C.i$ can be evaluated in the cache, given that $L.c$ can be probed successfully for ‘Berlin’.
- We have revealed that RCC $L.l \rightarrow A.p$ is actually redundant and thus need not be checked during cache loading or probing operations. We could also warn the user about this redundancy in his design, either when loading his complete specification into our caching system or in advance, when the user is designing his cache group assisted by a cache group adviser that implements our rules.
- Finally we have discovered four column-complete columns (among them admittedly two trivial ones): These promise more flexibility in choosing the cheapest probing strategy.

5 Conclusion

In this paper, we have presented four simple optimization rules that can be applied to a cache group after it has been designed. These rules do not touch the loading behavior, but make redundant information explicit that is contained in or derivable from the given cache group design. Furthermore, during optimization, unsafe cache groups can

be detected. This stock of information allows the cache manager to perform his tasks of loading, unloading, probing, and query evaluation more efficiently.

Alternatively, this information could be fed back interactively to the designer of a cache group to make him aware of the consequences of his decisions. Another type of information that would be useful in this setting is estimates about the loading costs of predicate extensions.

Our rules find the most useful redundant RCCs in situations that occur frequently. We have demonstrated which constellations in cache groups lie beyond the capabilities of our rules and how the rules could be extended to cope with those.

We have already implemented a DB-caching prototype called ACCache [6], which relies on our constraint-based caching model. It is realized on top of an existing relational DBMS and leverages its federated query execution capabilities. Within ACCache we can fill the cache; analyze, rewrite, and execute queries (partially) in the cache or in the backend DB; collect statistics about the usage of specific predicate extensions; and we can perform garbage collection based on these statistics. The making use of redundant RCCs and column-complete columns during this tasks is still to be added.

At the moment, we are developing an automated measurement environment, which will enable us to perform comparative benchmarks in order to assess quantitatively the actual benefit of our cache group optimization rules presented in this paper – among other aspects, such as the costs of loading and unloading predicate extensions or the overhead of probing, always in comparison to the lower latencies or reduced backend loads achievable.

References

1. Podlipinig, S., Böszörmenyi, L.: A survey of web cache replacement strategies. *ACM Computing Surveys* **35**(4) (2003) 374–398
2. Larson, P., Goldstein, J., Zhou, J.: MTCache: Transparent mid-tier database caching in SQL server. In: ICDE Conference, IEEE Computer Society (2004) 177–189
3. Altinel, M., Bornhövd, C., Krishnamurthy, S., Mohan, C., Pirahesh, H., Reinwald, B.: Cache tables: Paving the way for an adaptive database cache. In: VLDB Conference. (2003) 718–729
4. Härdter, T., Bühlmann, A.: Value complete, column complete, predicate complete – Magic words driving the design of cache groups. *VLDB Journal* (2006) Accepted for publication.
5. Scholl, W.: Cache-Group-Optimierung zur Effizienzsteigerung von Datenbank-Caches. Project thesis, TU Kaiserslautern (2006) <http://wwwdvs.informatik.uni-kl.de/pubs/DAsPAs/Sch06.PA.pdf>
6. Bühlmann, A., Härdter, T., Merker, C.: A middleware-based approach to database caching. In Manolopoulos, Y., Pokorný, J., Sellis, T., eds.: ADBIS 2006. Volume 4152 of LNCS., Thessaloniki (2006) 182–199

Construction of Tree-Based Indexes for Level-Contiguous Buffering Support

Tomáš Skopal, David Hoksza, and Jaroslav Pokorný

Charles University in Prague, FMP, Department of Software Engineering
Malostranské nám. 25, 118 00 Prague, Czech Republic

{tomas.skopal, david.hoksza, jaroslav.pokorny}@mf.f.cuni.cz

Abstract. In multimedia databases, the spatial index structures based on trees (like R-tree, M-tree) have been proved to be efficient and scalable for low-dimensional data retrieval. However, if the data dimensionality is too high, the hierarchy of nested regions (represented by the tree nodes) becomes spatially indistinct. Hence, the query processing deteriorates to inefficient index traversal (in terms of random-access I/O costs) and in such case the tree-based indexes are less efficient than the sequential search. This is mainly due to repeated access to many nodes at the top levels of the tree. In this paper we propose a modified storage layout of tree-based indexes, such that nodes belonging to the same tree level are stored together. Such a level-ordered storage allows to prefetch several top levels of the tree into the buffer pool by only a few or even a single contiguous I/O operation (i.e. one-seek read). The experimental results show that our approach can speedup the tree-based search significantly.

1 Introduction

The research in database indexing remains still a hot topic – its importance even increases with the emergence of new data types like multimedia data, time series, DNA sequences, etc. For such data, the tree-based indexes are often employed, e.g. the R-tree, X-tree, M-tree, and others [15], while apart from task-specific criteria of retrieval efficiency, the I/O costs still represent an important efficiency component. Simultaneously, the complexity of new data types makes them hardly indexable by tree-based structures, so the sequential search is often referred to perform better (in terms of I/O costs) than any tree-based index [20].

Despite the recent boom of new storage media (e.g. flash or hybrid disks), the best (and cheapest) medium for storage/indexing is still the magnetic hard disk drive (HDD) with rotating platters and moving heads. Due to its construction, the I/O efficiency of HDD depends on *access time* and *transfer rate*. The access time is determined by the *seek time* (head moves to a track), *settle time* (precise head positioning) and the *latency* (or rotational delay). The transfer rate is given by MB/s of sequentially (contiguously) read/written data from/to a track.

While HDD capacity doubles every year and transfer rate increases by 40%, the access time improves only by 8% (because of kinetic limitations of heads). Todays HDD can be of 300GB capacity, 50MB/s transfer rate and 10ms access

time. With 8KB disk blocks (or pages) used by file systems, the fetching of a block takes 10.16ms, so the access takes 98.5% of the total time. A contiguous fetch of 800KB data takes only 2.5x the time needed for fetching 8KB data. However, some two decades ago the HDDs exhibited different parameters, the access time about 100ms and the transfer rate at about 150KB/s. Thus, a random access to a disk block is relatively more expensive nowadays than some 20 years ago.

Sequential vs. Tree-based Indexing. The classic access methods have been developed based on a traditional disk model that comes with simplifying assumptions such as an average seek-time and a single data transfer rate. An excellent overview of these problems can be found in [19]. The B-tree or R-tree structures were developed in times of relatively cheap access costs (compared to the transfer rates). The tree node size (mapped to a block) was 2 or 4KB, while sequential reading of large data from HDD was not much cheaper than reading the data by multiple random-access retrievals, e.g. 7s vs. 32s in case of 1MB of data and 4KB blocks. By query processing, a traversal of 1/5 (or less) of the tree index sufficed to be faster than the simple sequential search. Today, the tree-based querying must traverse less than 1/86 to overtake the sequential search. Such a small proportion is achieved by B⁺-tree, or R-tree built on low-dimensional data. However, complex data cannot be retrieved in such an efficient way, because of their high dimensionality. Therefore, in modern applications the sequential search (or sequential-based indexes like VA-file [20]) is reported as more efficient (in terms of I/O costs) than indexing by tree-based structures.

How Large the Tree Nodes Should be? One can ask whether the access times could be reduced by enlarging the tree nodes. Then the number of nodes would be smaller and so the number of I/Os would decrease. Here the problem is in the increased number of entries stored in the node (the node capacity). Unlike B-tree, where the node split operation is of linear complexity with the number of entries, in R-tree or M-tree the complexity of node split is super-linear because of (sub)optimal partitioning of entries. A high node capacity also leads to worse approximations (e.g. MBRs in case of R-tree) in the parent node.

Second, although in B-tree the search in a single large node is fast because of use of interval halving, this is not possible in R-tree or M-tree where no universal ordering of entries is guaranteed. This has not to be critical in case of low-dimensional R-tree where the tuples-in-query testing is fast, however, in M-tree the sequential search within a node implies expensive distance computations.

1.1 Paper Contributions

In this paper we use level-separated buffering scheme which leads to more effective buffer utilization. Moreover, we introduce a modified split algorithm which keeps the tree index level-contiguous, that is, nodes belonging to a certain level in the tree are stored together. Such a modified index file layout allows to cheaply prefetch the top levels of the tree and thus further decrease the access costs.

2 Tree-Based Indexing

In this section we briefly summarize the structural properties of tree-based indexes and their secondary storage, including buffering into main memory. First, we assume "region-based" trees, where the data objects are stored in the leaves, while each entry in an inner node represents a (spatial) approximation of the appropriate subtree, e.g. R-tree's MBR, or M-tree's ball. We also assume an inner node with m entries (regions) has m children (see Figure 1(a)). Such assumptions are satisfied by R-tree, M-tree, but not by the B-tree (which is not region-based).

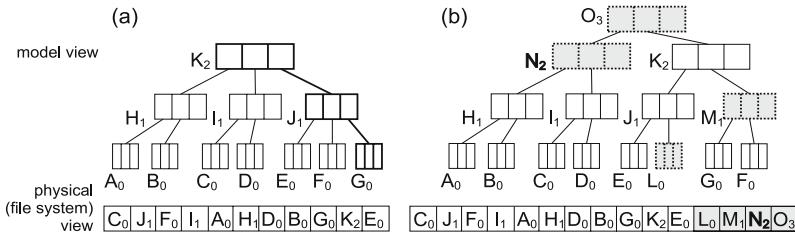


Fig. 1. (a) Insert into leaf G_0 . (b) The resulting tree, split up to the root.

We subscript a node by the number of its level (*level number*), starting by 0 at the leaf level (see Figure 1). Since indexes grow from bottom to top, a node's level number does not change. Besides the level number, each node obtains an identifier. A node is stored at address (or offset) in index file which is the identifier \times node size. The inner and leaf nodes are of single size (given in kilobytes).

Inserting and Splitting. By standard insertion, a leaf is found into which a new object is inserted. An overflowed leaf must be split between two leaves, one keeping the old identifier, and a brand new leaf. The two new entries describing two subtrees are inserted into the parent node (one entry is just updated). When the parent node overflows, the splitting is repeated (possibly up to the root level). In Figure 1, an insertion into the leaf G_0 raises a sequence of node splits.

Model Structure vs. Index File Layout. Note that the sequential ordering of nodes in the index file (physical view in Figure 1(a)) does not preserve the structure (the model view). This is because the new allocated nodes at the end of the index file come from different tree levels after a sequence of splits. In the optimal situation, the physical ordering exactly follows the model ordering given by breadth-first traversal of the tree. With such an organized index file we would be able to prefetch the neighboring nodes by a single contiguous read. Unfortunately, the standard splitting strategy cannot preserve the physical ordering of nodes in accordance with the model, because this would imply $O(n)$ insertion complexity (shifting many nodes), which is impracticable in most cases.

2.1 Standard Buffering and Prefetching

Like other database structures, also indexes use buffering [7] of blocks into memory frames. When a node is requested, its block is fetched from the disk and stored in a frame of the buffer pool. If the same node is requested later, it could still be in the buffer, so we avoid an I/O operation. Since the buffer is smaller than the volume of requested nodes, a *page-replacement policy* must be used, like LRU (LRU-T, LRU-P, LRU-K), MRU, FIFO, Clock, LFU, etc [15][13][14].

Because of reasons discussed in Section 2, we would like to access a large amount of data in single contiguous I/O operation. Instead of a single node, we could prefetch several additional nodes from the disk. Such prefetching is actually provided by the HW cache of the HDD. Unfortunately, the ordering of nodes in index file does not follow the tree structure. Hence, it would be inappropriate to force the prefetched nodes to be stored in the buffer, because such bulk-loading of nodes would lead to release of many nodes from the buffer which are (maybe) more likely to be requested than the prefetched ones.

3 Related Work

Typically, the tree-based indexes follow linear abstraction of HDD provided by file system. The only factor that has to be minimized is the number of random-access I/Os [8]. Most efforts in database indexing have been spent on improving filtering abilities with respect to the model (e.g. R-tree vs. X-tree [1] or M-tree vs. PM-tree [18]). Although the filtering improvements have a substantial impact on the overall efficiency (not only on the I/O costs), at some point further improving of the model is very hard. At that moment some lower-level techniques have to be investigated, related to HW and data storage issues.

3.1 Buffering Techniques

The I/O costs can be substantially reduced by appropriate buffering strategies. The classic work on index buffering [12] suggests the LRU replacement policy for B⁺-tree as the most effective. Also for multidimensional indexes the LRU policy has been proved as effective [6] (R-tree). In [11] a data-driven model has been proposed to predict the impact of buffering on R-trees. Moreover, specific replacement policies for spatial indexing have been proposed (suitable for R*-tree), where the nodes at the higher levels of a tree index are kept longer in the buffer [2].

3.2 Dynamic Layout Rearrangement

A general approach to speedup data retrieval is the dynamic rearrangement of storage layout [3][10]. The idea follows the assumption some access patterns are more frequent than other ones, so blocks belonging to the same pattern should be stored together to minimize movements of disk heads. The organ-pipe arrangement [16] is an example of such a layout. The rearrangement (also called

shuffling [16]) resembles file defragmentation for a specific access pattern, where the frequently accessed blocks are moved together during data retrieval with a hope this pattern will occur again. Although the rearrangement is a universal method for data management, its usage in database indexing is limited due to the absence of strong access patterns. Even if there exists an access pattern for a particular user, a multi-user access to the index will spoil the efforts spent by rearrangement because of many contradictory access patterns.

In our approach we use a kind of layout rearrangement, however, this one is performed during the construction of the index (i.e. not during query processing).

3.3 Physical Designs

Some recent works leave the linear abstraction of HDD and exploit physical properties of modern disks. Modern HDDs are manufactured with zoned recording, which groups adjacent disk cylinders into zones. Tracks are longer towards the outer portions of a disk platter as compared to the inner portions. Hence, more data can be recorded in the outer tracks when the maximum linear density is applied to all tracks. The results are multiple physical zones, where seek times and transfer rates vary significantly across the zones. In [21] the authors optimize dynamic multidimensional access methods (R*-tree) given a zoned disk model.

Another adjacent block utilization is presented in [17], however, the authors deal with storage of multidimensional data rather than indexing. The key idea is that HDD is, in fact, a three-dimensional medium where the adjacent tracks (either within a platter or within a cylinder) can be accessed efficiently.

The drawback of these methods is a requirement on specific system-level software, that provides applications with access to adjacent portions on the disk.

4 Level-Contiguous Indexing

Unlike the proposals in Section 3.3, we use the classic linear abstraction of data storage. Furthermore, we focus on indexes where complex queries are issued, i.e. queries where a substantial volume of nodes at the top levels must be processed (e.g. window or kNN query). Hence, we do not consider point or interval queries on B⁺-tree, since such queries result in simple one-path traversal. In other words, we consider an access pattern where the inner nodes are accessed much more frequently than the leaves. Based on the assumptions, we propose *level-contiguous* storage layout – an index storage partially preserving the model ordering of nodes for only a little construction overhead.

4.1 Index Traversal Analysis

In B⁺-tree, the most used query types are the point and interval queries defined for single-key domains, where the traversal is guided along a single path in the tree (an interval query must additionally search the leaf level), see Figure 2a.

Assuming that the queries are distributed uniformly, the probability that a node at a level of B⁺-tree will be accessed is inversely proportional to the number

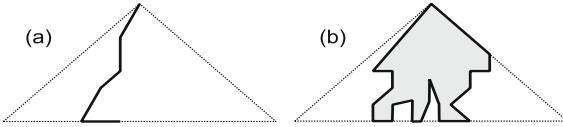


Fig. 2. (a) Point/interval search in B^+ -tree (b) Range/kNN search in R-tree or M-tree

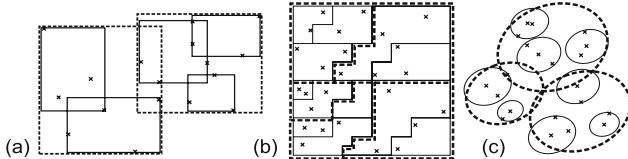


Fig. 3. Hierarchical space decomposition by (a) R-tree (b) UB-tree (c) M-tree

of nodes at the level, i.e. a leaf has the smallest probability and the root has 100%. However, some tree-based indexes are used for multidimensional or metric data, e.g. R-tree, X-tree, M-tree, where nodes represent regions in the indexed space. On such data there is no universal ordering defined, and also the query types are different. In particular, the R-tree is used for *range query* (or window query) and the M-tree is often used for *(k-)nearest neighbor (kNN) query*.

Since these structures index data which cannot be ordered, the tree traversal goes not along a single path. More likely, to reach all relevant data in the leaves, there must be multiple paths passed (see Figure 2b). The reason is that leaves relevant to a query are not clustered – they are spread over the entire leaf level.

Since the nodes represent regions in the indexed space, the top-level nodes' regions have large volume (they must cover all children regions, see Figure 3b). Then, during a query processing the nodes are checked against a query region and those children are further processed, which overlap the query. Obviously, the larger regions (nodes at the top levels) have greater probability to be accessed. With high-dimensional data, this means almost all top-level nodes are accessed (due to the *curse of dimensionality* [14]). Consequently, many random accesses are performed when querying high-dimensional data, so large portions of top levels are searched in randomized order. This is, in turn, often worse than contiguous sequential search of the entire index file.

4.2 Level-Contiguous Index Storage Layout

In our approach, we focus on "derandomization" of the I/O costs so that infrequent large contiguous I/Os are preferred over many random block I/Os. This can be achieved by a modification of index storage layout, in particular by ensuring that nodes are physically ordered by their level numbers (the order of nodes within a single level does not matter). In such a way, we can read all the nodes at several top levels by a single contiguous fetch, and store them into the buffer.

The idea makes use of adjusted node splitting. After an object has been inserted such that some node splits occurred, a special algorithm (called *SwapUp*, see Listing 1) is executed. The algorithm uses an array `mLevelStartIndex`, where its i -th entry stores the index file position of the first node belonging to i -th tree level. In principle, the algorithm propagates the new nodes (produced by splitting at the end of index file) in order to restore the ordering defined by level numbers. This is realized by swapping the new (misplaced) nodes with some old nodes which are located at first positions of a particular level in the index file.

Listing 1. (modified insertion algorithm, SwapUp algorithm)

```

method InsertObject(object o) {
    // insert o into the tree (this also involves splitting of overflowed nodes)
    ...
    // if some splits have occurred during the insertion, set splitCount = number of splits
    if (splitCount > 1) then SwapUp(splitCount)
}

method SwapUp(integer splitCount) {
    splitCount = splitCount - 1;
    for (i = 0; i < splitCount; i++) {
        integer swappedAtLevel = splitCount - i;
        for (j = 0; j < swappedAtLevel; j++) {
            SwapTwoNodesAt(mLevelStartIndex[i] + j,
                mLevelStartIndex[i] + GetNodesCountAtLevel(i) + j + 1);
        }
        mLevelStartIndex[i] = mLevelStartIndex[i] + swappedAtLevel;
    }
    if (splitCount == treeHeight) then { // treeHeight is the number of levels except the root level
        allocate mLevelStartIndex[splitCount];
        mLevelStartIndex[splitCount] = 0;
    }
}

```

Some notes: The `SwapTwoNodesAt` swaps the nodes defined by their identifiers (positions in index) together with both parent nodes' links pointing to the swapped nodes. To quickly access the parent node, a *parent identifier* must be additionally stored in each node. However, now also the parent identifiers of the child nodes of the two nodes being swapped must be updated. The `GetNodesCountAtLevel` returns the number of nodes at a given level before the insertion. Also note the *SwapUp* algorithm has not to be executed if just a leaf was split.

The algorithm running is explained in Figure 4a, which is index file layout related to the tree in Figure 1. Before insertion, the storage layout was level-ordered (see the white part in Figure 4a-1). After insertion, multiple splits caused ordering violation (see the grey part). The *SwapUp* algorithm now propagates the new nodes to correct positions. In Figure 4a-1, the new non-leaf nodes are swapped with the first 3 leaf nodes stored in the index. Then, the two remaining nodes are swapped with the first two level-1 nodes (see Figure 4a-2) and finally, the new root node O_3 is swapped with the old root K_2 (Figure 4a-3). The final index layout (let us denote it as *level-ordered index*) is presented in Figure 4a-4, where the top (bottom, respectively) arrows denote which parents (children) had to be updated with the new node's identifier during the swapping-up.

Time Complexity. Suppose n is the number of objects in the tree (i.e. $O(\log n)$ is the tree height). There are $O(\log n)$ seeks and contiguous data transfers (of

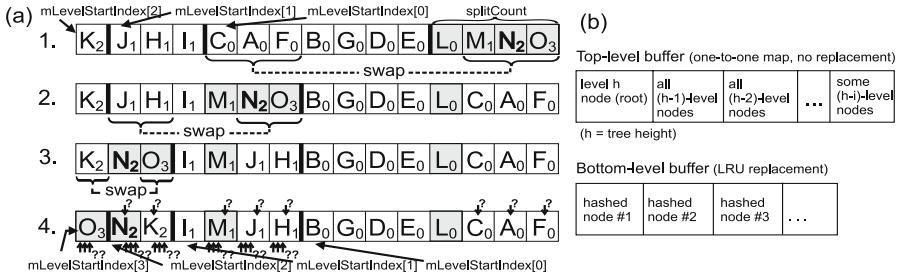


Fig. 4. (a) Swapping-up the new nodes after insertion (which caused multiple splits).
 (b) Top-level and Bottom-level buffer pools.

$O(\log n)$ blocks) performed during the swapping, while each of the $O(\log n)$ swapping steps spends $O(\log n)$ single-block I/Os on updating the links in parent/child nodes. Thus, the total worst-case complexity is $O(\log^2 n)$ when measured in block I/Os as well as in seek operations.

4.3 Level-Contiguous Buffering

As we have mentioned before, the nodes at top levels are the most accessed ones. It could appear that LRU/LFU replacement keeps the top-level nodes in buffer for a long time, since top-level nodes are considered as the recently (frequently) accessed ones. However, when a query is executed, the greatest amount of node reads belongs to the leaf level and the "valuable" top-level nodes are replaced by the leaves, because these are temporarily the most recently accessed ones.

Divided Buffer. Due to the obstacles caused by original LRU replacement in a single buffer pool, we use a kind of LRU-T policy – a buffer logically divided in two parts (see Figure 4b). The first part stores a user-defined number of top-level nodes (the *top-level buffer*), while once a node is loaded into top-level buffer, it will never be replaced. The second part behaves as an ordinary LRU-based buffer for the rest of nodes not buffered by top-level buffer (the *bottom-level buffer*).

Buffering the Top Levels. The top-level buffer can be populated either incrementally (by individual node requests) or by prefetching certain volume of the level-ordered index file. The prefetching itself can be accomplished in two ways. We can prefetch a large portion of the index at the beginning of index usage (*bulk prefetching*), so that the entire top-level buffer is populated. Or, we can prefetch smaller (yet sufficiently large) portions at the moment when a requested node is still not loaded in the top-level buffer (*incremental prefetching*). While the bulk variant minimizes the query time over many queries, the incremental one distributes the prefetch load among several queries.

5 Experimental Results

To prove the benefits of level-contiguous storage layout, we have performed experimentation with the R-tree and the M-tree. In the former case, the testing platform was a P4@3GHz, 1GB RAM, Maxtor OneTouch, Ultra ATA 133, 250GB@7200 rpm, 8MB cache, avg. seek<9.3ms, transfer rate 34 MB/sec. In the latter case we used P4@3.6GHz, 1GB RAM, Seagate Barracuda, SATA, 200GB@7200 rpm, avg. seek<8ms, 8MB cache, transfer rate 65MB/s. Both platforms were used with WinXP with disabled file-system cache (HDDs' HW caches were enabled for read), while both HDDs involved in tests were not system disks. In addition to R-tree and M-tree, we have also performed the tests on sequential file to set up a baseline, where for sequential query processing we have used a buffer of equal size as in case of the competitive R/M-trees. Most of the tests were executed for 100 different query objects and the results were averaged.

5.1 R-Tree Testbed

The first tests were aimed at indexing large synthetic multidimensional datasets by the R-tree and its level-contiguous modification (denoted as LC index in figures). There were 3 datasets generated, consisting of $3 \cdot 10^6$, $6 \cdot 10^6$, and 10^7 5-dimensional tuples. The tuples were distributed uniformly among 700, 800 and 1000 clusters, respectively. In Table 1 see the R-tree index characteristics.

Table 1. R-tree index statistics

Index size (4kB nodes): 160–511MB	Data objects: 3,000,000–10,000,000
Number of tree levels: 6–10	Node capacity: 92 in inners, 169 in leaves
Total buffer memory: 16,4MB (3.2–10,3% of index size)	Number of nodes: 7,679–19,723 inners, 31,545–104,810 leaves
LC construction time: 44min (for 3,000,000 dataset)	notLC constr. time: 27min (for 3,000,000 dataset)
Sequential file size: 82–245MB	Buffer for seq. file: 16,4MB (6.7–13,5%)

The number of disk accesses for window queries with increasing query selectivity (number of objects in the answer) is presented in Figure 5a. The label TopBuffer= $x\%$ denotes a bulk-prefetch index with size of top-level buffer equal to $x\%$ of all inner nodes (i.e. TopBuffer=0% means no top-level buffering, while TopBuffer=100% means all inner nodes can be buffered). The bottom-level buffer is maintained in the remaining buffer memory. As we can see, the LC index with TopBuffer=8% outperforms the classic R-tree ("notLC" indexes) as well as LC indexes with different TopLevel values. Note that we have utilized the top-level buffering also in the notLC indexes, however, here the top-level nodes cannot be prefetched, they were accessed one-by-one. In Figure 5b see the realtimes for the same situation. All the LC indexes show almost 100% speedup when compared to notLC indexes. Surprisingly, the LC indexes outperform the notLC indexes even in case that no top-level buffering and prefetching is employed. In Figure 5c the realtimes show behavior of LC/notLC indexes on the 10,000,000 dataset, and in Figure 6a see the disk accesses on the 3,000,000 dataset.

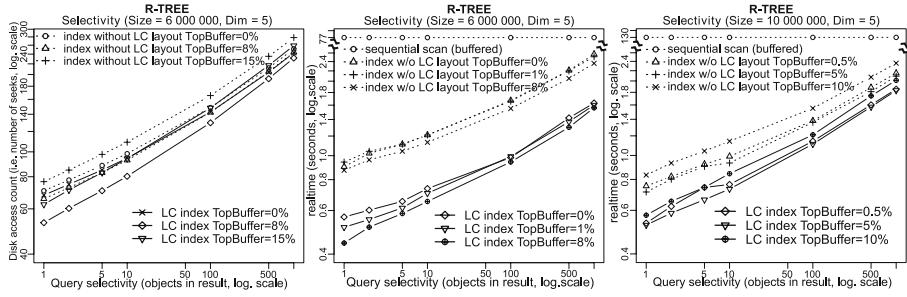


Fig. 5. R-tree: Disk accesses and realtimes for increasing query selectivity

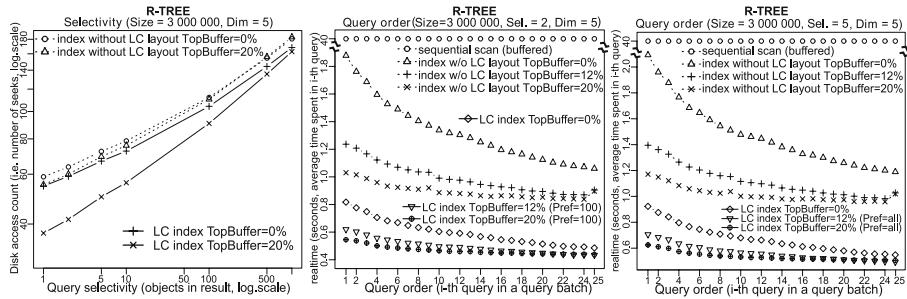


Fig. 6. R-tree: Disk accesses for increasing query selectivity and realtimes for typical response of i -th query in a query batch

We have also tested the impact of top-level buffering/prefetching with respect to the order of issued queries. In Figures 6b,c see the average realtime costs for queries with selectivity = 2 (5, respectively), according to the order of the query in a query sequence (or query batch). We can observe the benefits of LC indexes do not decrease in time. In Figure 6b the top-level nodes of LC indexes were prefetched incrementally, by 100 nodes, but as we can notice, there is no significant difference between prefetching incrementally or in a bulk (Figure 6c).

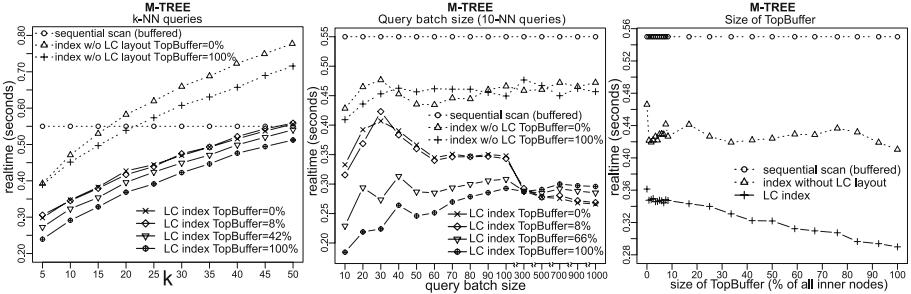
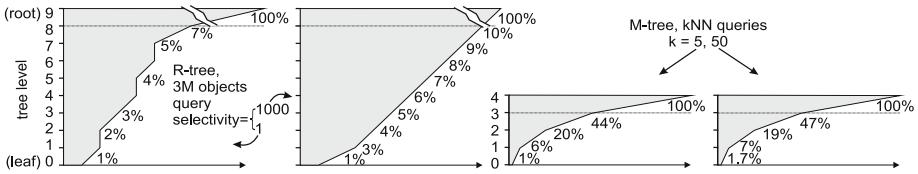
5.2 M-Tree Testbed

Second, we have implemented level-contiguous M-tree [5] and performed experiments with the Corel [9] feature vectors (65,615 images). The dataset consisted of 262,460 8-dimensional vectors, constructed by merging 4 feature representations (color and layout histograms, textures, color moments). The L_1 distance was used to measure image dissimilarity. See M-tree characteristics in Table 2.

In Figure 7a see the realtimes of kNN queries, with respect to increasing k . Although the classic notLC M-tree gets worse than the sequential file already at $k = 15$ (or $k = 20$ in case of M-tree with top-level buffering), the LC indexes remain efficient up to $k = 50$. The impact of query batch size is presented

Table 2. M-tree index statistics

Index size (2KB nodes): 29MB	Data objects: 262,460 (8D vectors)
Number of tree levels: 5 (root + 4)	Node capacity: 19 in inners, 29 in leaves
Total buffer memory: 2.9MB (10%)	Number of nodes: 1188 inners, 13180 leaves
LC construction time: 3.5min	notLC constr. time: 2.8min
Sequential file size: 9MB	Buffer for seq. file: 0.9MB (10%)

**Fig. 7.** M-tree: Realtimes for kNN queries depending on k , size of query batch, and proportion of TopBuffer**Fig. 8.** Structure of node accesses per level for queries in R-tree and M-tree

in Figure 7b, where the LC indexes do not deteriorate when compared with notLC indexes, they get even better. We have also examined the influence of top-level buffer proportion in the total buffer memory, see Figure 7c. We can observe that increasing volume of top-level buffer improves the realtimes quite significantly.

Finally, in Figure 8 see the structure of accesses to nodes per level in the tree-based indexes. Besides the root node, which must always be accessed, we can see that the nodes at top levels are accessed indeed frequently, especially in case of M-tree. Thus, the rationale for top-node buffering and level-contiguous layout seem to be well-founded, and we can expect level-contiguous layout could be beneficial also to other tree-based indexes, like X-tree, UB-tree and others.

In summary, the level-contiguous storage layout supports efficient utilization of access patterns usual for tree-based indexes, so that they can exploit the advantage of contiguous disk reading (like sequential search does it). This property dramatically reduces the random-access I/O overhead spent at top tree levels.

6 Conclusions

We have introduced level-contiguous storage layout for tree-based indexes. The new layout allows to prefetch the frequently accessed nodes at the top levels of any multidimensional or metric tree based on B^+ -tree. Moreover, we have used divided schema for level buffering, where the prefetched top-level nodes are stored separately and the replacement policies are not applied to them. The experimental results show that the prefetching together with the top-level buffering significantly improves the performance of query processing (up to 200% speedup) at the costs of a moderate increase of construction costs (about 30%).

Acknowledgments. This research has been supported by GAČR 201/05/P036 and GAČR 201/06/0756 grants provided by the Czech Science Foundation.

References

1. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
2. T. Brinkhoff. A Robust and Self-tuning Page-Replacement Strategy for Spatial Database Systems. In *EDBT*, pages 533–552, London, UK, 2002. Springer-Verlag.
3. S. D. Carson. A system for adaptive disk rearrangement. *Software - Practice and Experience (SPE)*, 20(3):225–242, 1990.
4. E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
5. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB'97*, pages 426–435, 1997.
6. A. Corral, M. Vassilakopoulos, and Y. Manolopoulos. The Impact of Buffering on Closest Pairs Queries Using R-Trees. In *ADBIS '01: Proceedings of the 5th East European Conference on Advances in Databases and Information Systems*, pages 41–54, London, UK, 2001. Springer.
7. W. Effelsberg and T. Haerder. Principles of database buffer management. *ACM Transactions on Database Systems (TODS)*, 9(4):560–595, 1984.
8. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
9. S. Hettich and S. Bay. The UCI KDD archive [<http://kdd.ics.uci.edu>], 1999.
10. H. Huang, W. Hung, and K. G. Shin. FS2: dynamic data replication in free disk space for improving disk performance and energy consumption. In *ACM SOSP '05*, pages 263–276, New York, NY, USA, 2005. ACM Press.
11. S. T. Leutenegger and M. A. Lopez. The Effect of Buffering on the Performance of R-Trees. *IEEE Transaction on Knowledge and Data Engineering*, 12(1):33–44, 2000.
12. L. F. Mackert and G. M. Lohman. Index scans using a finite LRU buffer: a validated I/O model. *ACM Transactions on Database Systems (TODS)*, 14(3):401–424, 1989.
13. R. Ng, C. Faloutsos, and T. Sellis. Flexible buffer allocation based on marginal gains. In *ACM SIGMOD*, pages 387–396. ACM Press, 1991.
14. E. J. O’Neil, P. E. O’Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *ACM SIGMOD*, pages 297–306, New York, NY, USA, 1993. ACM Press.

15. R. Ramakrishnan and J. Gehrke. *Database Management Systems, 3rd edition.* WCB/McGraw-Hill, 2003.
16. C. Rueemmler and J. Wilkes. Disk Shuffling, Technical Report HPL-CSP-91-30, Hewlett-Packard Laboratories, 1991.
17. S. W. Schlosser, J. Schindler, S. Papadomanolakis, M. Shao, A. Ailamaki, C. Faloutsos, and G. R. Granger. On multidimensional data and modern disks. In *4th USENIX Conference on File and Storage Technologies*, pages 225–238, 2005.
18. T. Skopal, J. Pokorný, and V. Snášel. Nearest Neighbours Search using the PM-tree. In *DASFAA '05, Beijing, China*, pages 803–815. LNCS 3453, Springer, 2005.
19. J. S. Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.
20. R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB '98*, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
21. B. Yu and S. Kim. An efficient zoning technique for multi-dimensional access methods. In *TEAA 2006, LNCS 3888, Springer*, pages 129–143, 2006.

A Workload-Driven Unit of Cache Replacement for Mid-Tier Database Caching

Xiaodan Wang¹, Tanu Malik¹, Randal Burns¹, Stratos Papadomanolakis²,
and Anastassia Ailamaki²

¹ Johns Hopkins University, USA

{xwang, tmalik, randal}@cs.jhu.edu

² Carnegie Mellon University, USA

{stratos, natassa}@cs.cmu.edu

Abstract. Making multi-terabyte scientific databases publicly accessible over the Internet is increasingly important in disciplines such as Biology and Astronomy. However, contention at a centralized, backend database is a major performance bottleneck, limiting the scalability of Internet-based, database applications. Mid-tier caching reduces contention at the backend database by distributing database operations to the cache. To improve the performance of mid-tier caches, we propose the caching of query prototypes, a workload-driven unit of cache replacement in which the cache object is chosen from various classes of queries in the workload. In existing mid-tier caching systems, the storage organization in the cache is statically defined. Our approach adapts cache storage to workload changes, requires no prior knowledge about the workload, and is transparent to the application. Experiments over a one-month, 1.4 million query Astronomy workload demonstrate up to 70% reduction in network traffic and reduce query response time by up to a factor of three when compared with alternative units of cache replacement.

1 Introduction

The sciences are collecting and analyzing vast amounts of observational data. In Astronomy, cataloging and mapping spectral characteristics of objects in only a fraction of the sky requires several terabytes of storage. Data are made available to remote users for processing, for example through SkyQuery [1], a federation of Astronomy databases and part of the World-Wide Telescope [2]. However, SkyQuery faces an impending scalability crisis. The federation is expected to expand from roughly a dozen members today to over a hundred in the near future [3]. Furthermore, member databases, such as the Sloan Digital Sky Survey (SDSS) [4], are accumulating data at an astonishing rate.

Mid-tier caching is an attractive solution for increasing scalability, availability, and performance of distributed database applications [5]. We study mid-tier caching in the context of SkyQuery using bypass-yield caching [6]. Bypass-yield caching replicates database objects, *e.g.* columns (attributes), tables, or views, at caches deployed near the clients so that queries are served locally, reducing network bandwidth requirements. Caches service some queries in cache and ship other queries to be evaluated at the backend database.

Our experience with bypass-yield caching indicates that query evaluation performance in the cache is also critical. Despite the network benefits, poor I/O

performance in caches may result in inferior overall performance. Mid-tier caches lack the indices that are vital to I/O efficiency in databases. Maintaining indexes in a cache is prohibitively expensive given that (1) index construction is time consuming and I/O-intensive, (2) cache data are continuously changing, and (3) indices consume space, polluting the cache with replicated data. In this paper, we extend previous work on network traffic reduction with bypass-yield caching [6] by exploring ways to simultaneously improve query execution performance in the cache.

In existing mid-tier caching models, the storage organization employed by the cache is either tied to the backend database or defined *a priori*, e.g. columns [5], tables [5], vertical or horizontal fragments of base tables [7][8], or views [9]. Our work differs from previous caching approaches in two ways. First, we explore *dynamic* cache storage organizations that take into account workload information to improve query performance. Second, we evaluate alternative units of cache replacement in terms of their network traffic reduction benefits.

We propose a workload-driven technique for choosing the unit of cache replacement that is adaptive and self-organizing. Our model employs query prototypes in which each prototype is a combination of attributes that is accessed by the same class of queries. Prototypes serve as the logical unit of cache replacement. Query prototypes are adaptive in that prototypes are defined dynamically based on the access pattern of queries that appear in the workload. This is useful for scientific databases in which an *a priori* workload is not available. In particular, Astronomers are constantly finding new experiments to conduct in SkyQuery, making it difficult to identify a static set of frequently accessed database objects. Query prototypes are self-organizing in that changes to the storage organization are part of the cache replacement decision. Each prototype is optimized for a specific class of queries and, as workloads change, the storage layer changes accordingly. This makes it unnecessary to reorganize the cache contents periodically to improve query performance.

Our experiments show that query prototypes result in a factor of three reduction in query response time when compared with caching of columns, tables, and vertical partitions of backend tables. Prototypes also exhibit low cache pollution and high network savings. This is especially true at low cache sizes in which 40% less network traffic was generated when compared with the next best method.

We emphasize that this paper does not introduce a new caching algorithm, but presents a technique for specifying the unit of cache replacement that improves performance without sacrificing the inherent merits of mid-tier database caching.

2 Caching for Scientific Databases

We briefly describe the framework used to study our approach and explain why choosing the unit of cache replacement is relevant to database caching.

2.1 Cache Environment

SkyQuery [1], a federation of Astronomy databases, is a Web-based application in which caching drastically reduces network traffic. In SkyQuery, Web mediators or portals located in close proximity to the users serve as the interface between user queries

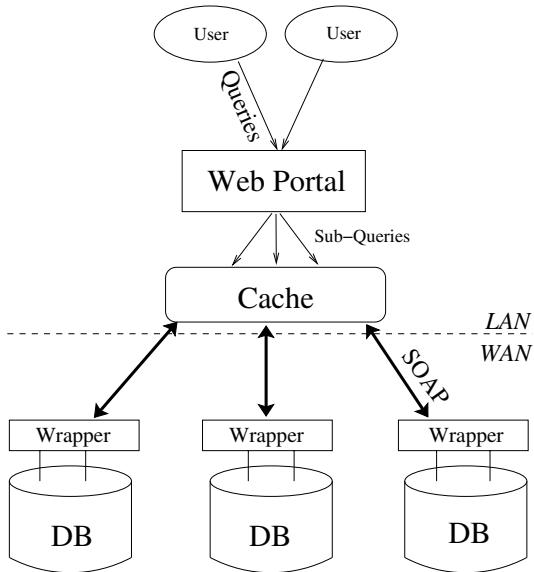


Fig. 1. Mid-tier database caching in SkyQuery

and member databases in the federation. As shown in Figure 1, the portal communicates with member databases via a wrapper interface. The wrapper interface allows member databases to remain autonomous and heterogeneous entities. We are currently building a cache prototype in SkyQuery in which the cache resides at the portal and utilizes the wrapper interface to transfer data, process queries, and collect schema information.

The SkyQuery workload is read-only and contains a rich variety of range, aggregate, identity, and spatial queries submitted through a large community of Astronomers. User queries either execute locally in the cache or are shipped to backend databases. However, executing queries at the backend database generates a lot of network traffic over WAN, because query results are transferred back to the user. The goal is to cache database objects so that most of the data transfer is from the cache to the user over LAN.

We employ bypass-yield caching (BYC) [6], which was developed for SkyQuery. The primary goal of caching in this environment is network traffic reduction because queries are network-bound [6]. BYC is an economic framework in which network bandwidth is the currency and network traffic reduction is the goal. The decision to ship a database object to the cache represents an investment in which the cost to load an object is recovered through expected network savings. Queries that access objects which fail to yield positive network savings in this economic model are bypassed, *i.e.* shipped to the backend database for execution. BYC is also flexible with respect to the unit of cache replacement; replacement can be performed on individual columns, tables, or tuples.

Query prototypes is an extension to BYC by making storage management part of the cache replacement decision. Storage management is important because mid-tier caches operate in index-free environments in which cache data are continuously changing and cache space is constrained. While BYC identifies data that are beneficial to the cache, it

does not consider how data layout on disk impacts query execution performance. Query prototypes not only capture data that are useful to cache but also how data should be organized on disk.

2.2 Choosing the Unit of Cache Replacement

The granularity of cache replacement has a significant impact on the overall network performance [6]. Network performance is governed by the utilization of cache space. In general, caching objects at too fine a granularity increases maintenance overhead. For example, if data granularity is chosen at the level of individual tuples, then significant cache space is needed to maintain the relationships among all of the cached tuples. On the other hand, too coarse a granularity degrades cache utilization by emptying a large portion of the cache during object replacement [10].

Query performance is governed by the effective clustering of data in the cache. The best clustering is obtained if groups of workload-related tuples are cached, as in semantic caching. However, semantic caching requires that workloads exhibit query locality, which is not true of Astronomy workloads [6]. Furthermore, splitting, coalescing, and containment checking is difficult when workloads consist of complex queries, and not just range queries [7]. Thus, from the perspective of query performance, cache replacement must be performed at the granularity of tables, columns, or vertical or horizontal fragments of the backend database. However, these database objects are defined during the database design process, which is concerned with eliminating redundancy and update anomalies in addition to workload access patterns. Naively choosing any of the above as the choice of caching granularity forces columns that are logically related but rarely accessed together to be stored together in the cache. This hurts both query and network performance. In contrast, if the unit of cache replacement is adaptive so that columns are grouped in a manner that reflects changing query access patterns, then overall cache performance is improved.

3 Related Work

In this section, we summarize work on mid-tier caches that define static and dynamic units of cache replacement. We also review several database design methods for improving query performance, including vertical partitioning and materialized views.

3.1 Statically-Defined Cache Replacement

Mid-tier caching provides greater scalability and availability, increased performance, and quality of service guarantees [5]. Several high performance mid-tier caching systems allow for flexibility in defining the unit of cache replacement [7, 8, 9]. Cache Tables [7] allows for the caching of *declarative cache tables*, which corresponds to a table, column, or materialized view from the backend database. Similar flexibility in the unit of cache replacement is achieved in TimesTen [9] through the definition of *cache groups* and MTCache [8] through the use of *select-project views*. However, the unit of cache replacement in these systems is specified *a priori* during initialization. Also, the unit of cache replacement is static and does not adapt to workload changes. We found that

adapting the storage organizations of cached contents by changing the unit of cache replacement over time significantly improves query performance.

3.2 Dynamically-Defined Cache Replacement

Predicate-based or semantic caching supports an adaptive unit of cache replacement [11]. Caching is performed on groups of spatially related tuples, as defined by a set of predicate conditions, which exhibit high semantic reuse. Unlike statically-defined caching schemes, which are susceptible to poor clustering, semantic regions can grow or shrink in size to adapt to workload changes. Judicious data placement, accomplished by preserving spatial locality of data that are frequently accessed together, improves query performance significantly over static schemes [10].

While semantic caching is attractive, there are some drawbacks. Maintaining a semantic description of the data is feasible when workloads consist mostly of simple *select-project-join* queries [7]. Also, workloads should exhibit semantic reuse in which data items contained in the result of a query are later reused. These properties allow for efficient checking of query containment against cached data [11]. However, Astronomy workloads comprise of nested queries with user-defined functions and complex joins, which are unsuitable for semantic caching. More importantly, scientific workloads exhibit little semantic reuse but frequent syntactic reuse [6]; *i.e.* specific data items experience little to no reuse, but queries request data from the same group of columns. We exploit syntactic reuse by caching dynamically-defined groups of columns.

3.3 Database Design Methods

Vertical partitioning identifies I/O efficient placement of columns at the storage layer to improve query performance [12][13][14][15][16][17]. Early works introduced the notion of affinity, the frequency in which attributes are accessed together, to evaluate the placement of columns [12][13][18]. Data columns are grouped together by applying a clustering algorithm on affinity values. Recent work suggests that affinity value is decoupled from actual I/O cost and is a poor predictor of query performance. They propose more sophisticated cost models that estimate the I/O cost of performing database operations [14][15]. AutoPart [15] is a workload-based, automated database design tool that improves query performance through vertical and horizontal partitioning. While query prototypes can be described as a vertical fragmentation of the database, it does not partition the database algorithmically. Also, prototypes are not disjoint in that multiple prototypes may replicate the same data column.

Materialized views allow for arbitrary vertical and/or horizontal fragmentations of base tables. Materialized views are concrete tables derived from underlying base relations to enable I/O efficient accesses. In ViewCache, a framework for managing materialized views is provided that balances storage overhead with performance [19]. Views are made compact by storing pointers to records in the base relation and are materialized on demand. However, ViewCache does not specify the appropriate views to create.

Multi-query optimization, a technique used in data warehouse systems, provides one solution to view selection [20]. The goal is to exploit shared data between a set of queries or views and identify additional views for transient or permanent materialization that are used to share intermediate results and improve performance. Multi-query

optimization has been successfully applied in view maintenance [21] and in the optimization of inter-query execution [22]. However, evaluating shared sub-expressions increases the complexity of query optimization. Query prototype caching does not increase optimization cost because each query is evaluated against a single prototype. Moreover, multi-query optimization is applied to known workloads that are fairly static and that have high overlap across queries. Neither are assumptions for query prototypes.

4 Query Prototype Caching

This section provides a formal description of query prototypes. The method for specifying query prototypes takes as input a set of queries \mathcal{Q} and outputs a set of prototypes \mathcal{P} , which serve as the unit of cache replacement. Each query is matched against exactly one prototype, whereas each prototype is derived from a set of related queries.

4.1 Definition

Let $\mathcal{R} = R_1, \dots, R_n$ be the set of all tables at the backend database. Each table consists of a set of attributes. Let $\mathcal{A} = A_{11}, \dots, A_{1m}, A_{21}, \dots, A_{np}$ as the set of all attributes at the backend database in which A_{ij} is the j^{th} attribute in relation R_i .

Let \mathcal{Q} be the set all queries in the workload in which $q_i \in \mathcal{Q}$ is the i^{th} query in the workload. AutoPart [15] introduced the concept of a *Query Access Set* (QAS), which is the subset of attributes from a *single* relation in \mathcal{R} that are referenced by a query in \mathcal{Q} . For query prototypes, we redefine *Query Access Set* to be the set of attributes from every relations in \mathcal{R} that are referenced by a query in \mathcal{Q} .

Let $\text{QAS}(q_i, \mathcal{A})$ be defined as the set of attributes from \mathcal{A} that are referenced by query q_i . For query prototype caching, we consider queries q_i and q_j to be equivalent if and only if $\text{QAS}(q_i, \mathcal{A}) = \text{QAS}(q_j, \mathcal{A})$ – that is, they access the same set of attributes. A set of queries, in which the QAS of these queries are identical, make up an equivalence class in the workload. Each prototype in \mathcal{P} represents a unique equivalence class in the workload. Thus, to cache a prototype P_k , the set of attributes referenced by queries in P_k are loaded into the cache as one unit.

We demonstrate the concept of query prototypes using three queries derived from an Astronomy workload [4]:

- q_1 : SELECT objID FROM Galaxy, SpecObj
WHERE objID = bestobjID and specclass = 2 and z between 0.121 and 0.127
- q_2 : SELECT objID, ra, dec FROM PhotoPrimary WHERE dec between 2.25 and 2.75
- q_3 : SELECT top 1 ra, dec FROM PhotoPrimary WHERE objID = 5877311875315

The example shows two unique query prototypes in which the *Query Access Set* for q_2 and q_3 are identical:

$$\begin{aligned}\text{QAS}(q_1, \mathcal{A}) &= \{\text{Galaxy:objID}, \text{SpecObj:bestobjID}, \text{SpecObj:specclass}, \text{SpecObj:z}\} \\ \text{QAS}(q_2, \mathcal{A}) &= \{\text{PhotoPrimary:objID}, \text{PhotoPrimary:ra}, \text{PhotoPrimary:dec}\}\end{aligned}$$

4.2 Discussion

To materialize a prototype in the cache, the group of attributes belonging to the prototype is allocated a unique set of relation files in the storage layer. Spatial locality among attributes belonging to each prototype is preserved because storage is not shared between prototypes with overlapping attributes. Queries serviced from the cache are matched against a prototype and are rewritten to address relations in the cache. Furthermore, loading or evicting a prototype simply requires that the corresponding relations be added or dropped from the database.

Two properties are worth noting for query prototypes. Prototypes are rarely disjoint; attributes appear in multiple prototypes. This introduces attribute replication because prototypes that overlap do not share storage. A prototype can also contain attributes from multiple tables. One option for storing attributes from multiple tables computes the cross product and stores the result in a single table. However, this utilizes cache storage unwisely, because the storage required scales exponentially with the number of joins [23]. Instead, we store attributes belonging to different backend relations in separate tables and compute the join during query execution.

Theoretically, the number of query prototypes can be very large, equal to $\min(\# \text{ of queries}, 2^n)$ where n is the number of attributes referenced by the workload). In practice, query prototypes provide a compact summary of the workload, even those with millions of queries. We found that much of the science is conducted through prepared SQL queries via form-based applications or custom scripts that iterate over the database by accessing the same combination of attributes. This observation was exploited in a work on active, form-based proxy caching for backend databases [24]. Luo and Naughton acknowledged that while caching for queries with arbitrary SQL syntax is difficult, queries submitted through HTML forms exhibit only a handful of distinct types. The semantic information from these queries is captured through a compact set of *templates*, which are parameterized query specification in which parameter values are provided at runtime. Our approach applies to general queries and does not distinguish between form-based and arbitrary user queries.

4.3 Performance Implications

Query prototypes improve query performance at the cache by reducing the amount of data read from disk. Since prototypes contain only the attributes referenced by a query, accessing them is much faster than accessing the entire base table. Query prototypes reduce scan costs by ensuring that only data from columns requested by a query are read from disk. Also, each query is executed against a single prototype in which the number of join operations is never more than the number of backend tables a query accesses.

Besides improving performance for the queries executed in the cache, query prototypes perform well in terms of network bandwidth. Contrary to using entire tables as the cache replacement unit, query prototypes avoid unnecessary transfers of attributes that are not referenced by the workload. Although single-column partitions also have this property [6], they suffer from the *file-bundling* effect [25] in which evicting a single column from a group of columns that are accessed together renders the cache ineffective for the entire group.

Cache pollution is the loading of redundant or unused database objects that adversely impact performance by evicting useful objects. Pollution limits the applicability of query prototypes to general workloads. Since prototypes are not disjoint, the cache allocates storage for duplicate columns. If the number of prototypes is too high, then no single prototype serves enough queries to provide positive network savings. If the overlap between prototypes is high, then attribute replication quickly pollutes the cache.

5 Experiments

Our experiments use a one-month query trace from the Sloan Digital Sky Survey (SDSS), a major site in SkyQuery [4]. The trace consists of 1.4 million read-only, SQL queries generating nearly 360GB of network traffic against Data Release 4 (DR4), which is a two-terabyte database. To finish I/O experiments in a reasonable time, we take a ten percent sample of the DR4 database, roughly 200GB in size.

We evaluate query prototypes against three units of cache replacement: columns, tables, and logical groupings of columns as determined through vertical partitioning. For column caching, we store related columns in the same table rather than storing each column in a separate table. Query performance in the latter approach is disastrous using row-oriented, relational databases.

We adapt existing vertical partitioning algorithms to caching. Traditionally, partitioning takes as input a representative set of queries and outputs an alternative, I/O efficient database schema. Thus, partitioning algorithms are designed for a different set of goals than caching – that is, improving query performance rather than network usage. Nonetheless, we expect that the same technique for improving spatial locality among columns that are accessed together will group columns that are relevant to the cache. We choose AutoPart [15], a high-performance vertical partitioning algorithm, for our experiments. To ensure that the unit of cache replacement is adaptive, we periodically update the column groupings by rerunning the algorithm with new queries. We also restrict input to the algorithm to queries with results sizes greater than one megabyte because it is not economical to cache for queries with small result sizes.

All experiments use the DR4 database running on Microsoft’s SQL Server 2000. Our main workstation is a 3GHz Pentium IV machine with 1GB of main memory and two SATA disks. SQL Server uses one disk for logging and stores the database on a second, 500GB disk.

5.1 Query Workload

Upon analyzing the DR4 trace, we discover that query prototypes provide a compact representation of the workload. The entire 1.4 million trace consists of only 1176 prototypes, owing to schema reuse in which a limited combination of columns are repeatedly accessed. Moreover, a handful of prototypes capture most of the workload. 11 unique prototypes describe 91% of the entire workload while 6 unique prototypes generate 89% of the network traffic. Query prototypes that occur frequently do not correspond to those that generate most of the network traffic. The latter dictate cache replacement decisions whereas the former do not.

5.2 Query Performance

We measure query performance by deploying the cache for the sampled database and using the entire 1.4 million query workload as input to the cache. The cache size is set at 1% of the sampled database. Using a small relative cache size is appropriate, because caches are likely to have only a fraction of the several terabytes of storage available to backend databases. Figure 2 shows the performance of queries executed at the cache. Caching query prototypes results in the best performance with an average query response time of 474ms, which is up to three times faster than other strategies.

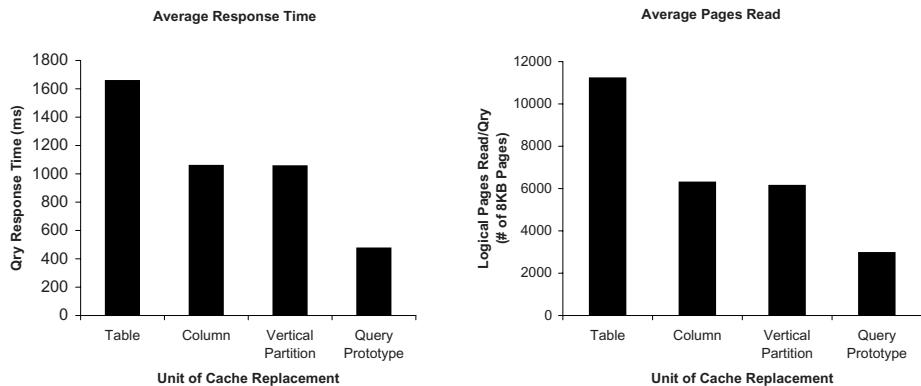


Fig. 2. Query performance by unit of cache replacement

The lack of prior knowledge about the workload limits the effectiveness of vertical partitioning. The partitioning algorithm takes all high-yield queries from the workload as input, but only a fraction of those queries are executed in the cache. This results in a mismatch between the workload executed in the cache and the workload provided as input to the partitioning algorithm.

We also measure logical database pages read to compare sequential access performance. Unlike physical reads, buffering and fragmentation on disk do not affect logical reads. The trend is consistent with query response times; query prototypes employ the most I/O-efficient layout of data and incur the fewest page reads.

5.3 Network Savings

In Figure 3, we compare network performance on the un-sampled database for different caching strategies at various increments of cache size. Query prototypes outperform column caching at cache sizes lower than 2% of the database, while table caching lags far behind. The enormous size of each table means that for cache sizes less than 0.5%, only a handful of objects fits in cache when table caching is used. As cache size increase, column caching exhibits a slight advantage over query prototypes. With the cache

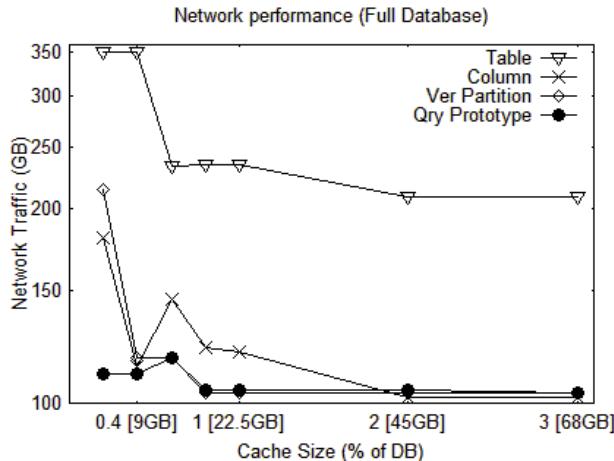


Fig. 3. Network traffic by unit of cache replacement. Network cost without caching is 357GB

at 3% of the database, query prototype caching generates 104GB of network traffic versus 102GB for column caching. Vertical partition caching also performs well and do a good job of grouping columns that provide positive network savings.

A conspicuous feature at cache sizes lower than 2% is that query prototype outperforms column caching by a large margin, generating up to 40% less traffic. This is explained by the file-bundling effect [25]. Specifically, column caching makes poor replacement decisions at low cache sizes when cache resident times are shorter and object evictions are more frequent. The resulting mix of columns in the cache have a lower probability of satisfying incoming queries.

5.4 Cache Pollution

In query prototypes, pollution arises from attribute replication. For table and vertical partition caching, columns that do not provide any network savings pollute the cache because they are grouped with columns that yield positive network savings.

Figure 4 shows cache pollution on the un-sampled database for different caching strategies. Attribute replication for query prototypes remains at 5% or lower for the most part. There is a sharp rise at 0.6% cache size, which does not significantly impact network savings because cache space is not fully utilized. Immediately after is a sharp drop because prototypes that were previously too large to fit into the cache are loaded. This displaces several smaller prototypes, resulting in fewer, large prototypes. Pollution remains fairly stable afterwards as the number of cached prototypes increases. Caching vertical partitions exhibit significantly more pollution than query prototypes. This is because partitioning algorithms are not designed for caching and do not always separate attributes that provide network savings from those that do not.

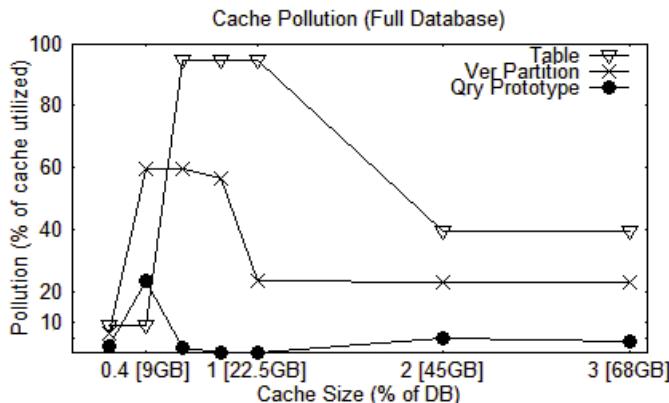


Fig. 4. Cache pollution, as a percent of cache space utilized, by unit of cache replacement

6 Conclusions and Future Work

We present a workload-driven approach to specifying the unit of cache replacement for scientific workloads that is adaptive and self-organizing. Our experiments illustrate that query prototypes achieve superior query and network performance with little cache pollution. However, prototypes are susceptible to cache pollution for workloads with high attribute overlap. To address this, we are considering merging prototypes with shared attributes in order to strike a balance between query performance and cache pollution. Vertical partition caching is promising because unlike query prototypes, it is not susceptible to a high degree of attribute replication. However, directly adapting existing partitioning algorithms to caching is unsuitable because these algorithms optimize query execution cost but do not consider network costs. We will extend partitioning algorithms to optimize for multiple costs in the future. Finally we plan to extend query prototypes to support updates.

References

1. Malik, T., Szalay, A.S., Budavri, A.S., Thakar, A.R.: SkyQuery: A Web Service Approach to Federate Databases. In: CIDR. (2003)
2. Gray, J., Szalay, A.: Online Science: The World-Wide Telescope as a Prototype for the New Computational Science. Presentation at the *Supercomputing Conference* (2003)
3. Szalay, A., Gray, J., Thakar, A., Kuntz, P., Malik, T., Raddick, J., Stoughton, C., Vandenberg, J.: The SDSS SkyServer - Public Access to the Sloan Digital Sky Server Data. In: SIGMOD. (2002)
4. The Sloan Digital Sky Survey. <http://www.sdss.org>
5. Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B.G., Naughton, J.F.: Middle-tier Database Caching for E-Business. In: SIGMOD. (2002)
6. Malik, T., Burns, R., Chaudhary, A.: Bypass Caching: Making Scientific Databases Good Network Citizens. In: ICDE. (2005)

7. Altinel, M., Bornhvd, C., Krishnamurthy, S., Mohan, C., Pirahesh, H., Reinwald, B.: Cache Tables: Paving the Way for An Adaptive Database Cache. In: VLDB. (2003)
8. Larson, P., Goldstein, J., Guo, H., Zhou, J.: MTCache: Mid-Tier Database Caching for SQL Server. In: ICDE. (2004)
9. The TimesTen Team: Mid-tier Caching: The TimesTen Approach. In: SIGMOD. (2002)
10. Dar, S., Franklin, M.J., Jonsson, B.T., Srivastava, D., Tan, M.: Semantic Data Caching and Replacement. In: VLDB. (1996)
11. Keller, A.M., Basu, J.: A Predicate-based Caching Scheme for Client-Server Database Architectures. VLDB (1996)
12. Hammer, M., Niamir, B.: A Heuristic Approach to Attribute Partitioning. In: SIGMOD. (1979)
13. Navathe, S., Ceri, S., Wiederhold, G., Dou, J.: Vertical Partitioning Algorithms for Database Design. ACM Trans. Database Syst. **9**(4) (1984) 680–710
14. Chu, W.W., Ieong, I.T.: A Transaction-Based Approach to Vertical Partitioning for Relational Database Systems. IEEE Trans. Softw. Eng. **19**(8) (1993) 804–812
15. Papadomanolakis, S., Ailamaki, A.: AutoPart: Automating Schema Design for Large Scientific Databases Using Data Partitioning. In: SSDBM. (2004)
16. Cornell, D.W., Yu, P.S.: An Effective Approach to Vertical Partitioning for Physical Design of Relational Databases. IEEE Trans. Softw. Eng. **16**(2) (1990) 248–258
17. Agrawal, S., Narasayya, V.R., Yang, B.: Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design. In: SIGMOD. (2004)
18. Navathe, S.B., Ra, M.: Vertical Partitioning for Database Design: A Graphical Algorithm. In: SIGMOD. (1989)
19. Roussopoulos, N.: An Incremental Access Method for ViewCache: Concept, Algorithms, and Cost Analysis. ACM Trans. Database Syst. **16**(3) (1991) 535–563
20. Sellis, T.K.: Multiple-Query Optimization. ACM Trans. Database Syst. **13**(1) (1988) 23–52
21. Mistry, H., Roy, P., Sudarshan, S., Ramamritham, K.: Materialized View Selection and Maintenance Using Multi-Query Optimization. In: SIGMOD. (2001)
22. Roy, P., Seshadri, S., Sudarshan, S., Bhobe, S.: Efficient and Extensible Algorithms for Multi Query Optimization. In: SIGMOD. (2000)
23. Ioannidis, Y.E., Christodoulakis, S.: On the Propagation of Errors in the Size of Join Results. In: SIGMOD. (1991)
24. Luo, Q., Naughton, J.F.: Form-Based Proxy Caching for Database-Backed Web Sites. In: VLDB. (2001)
25. Otoo, E., Rotem, D., Romosan, A.: Optimal File-Bundle Caching Algorithms for Data-Grids. In: ACM/IEEE Supercomputing (SC). (2004)

J⁺-Tree: A New Index Structure in Main Memory*

Hua Luan^{1,2}, Xiaoyong Du^{1,2}, Shan Wang^{1,2}, Yongzhi Ni^{1,2}, and Qiming Chen³

¹ School of Information, Renmin University of China, Beijing, China

² Key Laboratory of Data Engineering and Knowledge Engineering, Renmin University of China, MOE, Beijing, China

³ HP Labs China, Beijing, China

{luanhua, duyong, swang, ni}@ruc.edu.cn, qiming.chen@hp.com

Abstract. As the memory capacity increases and the hardware becomes cheaper, main memory databases (MMDB) have come true and been used in more and more applications, because they can provide better response time and throughputs. The advent of MMDB requires a reconsideration of data structures and algorithms of traditional DBMS. The index structure is one of the most important aspects that need be redesigned since it can affect the overall system performance heavily. Even though the T-tree index, which was proposed for main memory databases, has been widely accepted as a promising index structure. B⁺-tree and its variants still have their advantages in memory and are also regarded as the potential main memory database index structures. In this paper, we propose a new indexing technique called J⁺-tree for MMDB, inspired by the Judy structure which is an associative array data structure. Our J⁺-tree index not only holds the advantages of Judy (such as good single value search characteristic) but also outperforms it in many ways. For example, J⁺-tree can obtain better performance for range queries that are very slow in Judy structure. We compare the J⁺-tree index with Judy, T-tree and B⁺-tree on time and space aspects, and the experimental results show that J⁺-tree can provide better overall performance in main memory.

1 Introduction

Nowadays, computers with main memory size in the order of magnitude of gigabytes are available. With the advent of larger and cheaper memory, main memory databases (MMDB) where the whole database or large portions of it can fit in memory have gained more attention of database researchers and the industry. It is believed that MMDB can provide better performance as compared to traditional disk resident databases [4] [7]. There have existed many main memory database systems, from prototype implementations - Starburst system from IBM

* This work is partly supported by a grant from HP Lab China, NSFC China No.60573092, NSFC China No.60496325, China Grid No.CNGI-04-15-7A and NSFC China No.60503038.

Almaden Research Center [7] and Dali system from the AT&T Bell Laboratories [10] to commercial systems - TimesTen system purchased by Oracle Corporation in 2005 [11] and Altibase system of Altibase Corporation in Korea [5].

In main memory databases, indexes are still needed to access data fast and are useful for single value queries, range queries and indexed nested looping joins. The index structure is one of the primary factors that affect the overall system performance heavily, and has become an important research issue in MMDB. T-tree [6], proposed by Lehman et al, has been widely used as a major MMDB index structure - it is adopted by all the database systems mentioned above. However, some researchers argue that B⁺-tree and its variants such as CSB⁺-tree can provide better CPU cache behavior than T-tree [8] [9], so they can also be regarded as main memory index structures, although B⁺-tree is originally designed for conventional disk resident database systems.

The T-tree and B⁺-tree indexes have the same feature - the depth of the trees increases as the number of keys gets larger. The search cost, which is also part of update cost, is in proportion to the depth of the tree. So when a large number of keys need be stored in T-tree or B⁺-tree, which is common in main memory databases, their search cost is much expensive. In this paper, we propose a new index structure called J⁺-tree, inspired by a smart digital tree named Judy [1]. A Judy tree with 32-bit keys has a maximum of four levels and eight levels for 64-bit keys. Our J⁺-tree retains this good structure characteristic and has a low search cost - in spite of the number of keys, there are at most five levels in J⁺-tree when keys are 32-bit and nine levels for 64-bit keys.

Unlike Judy which keeps the whole keys in a Judy tree, J⁺-tree stores all the keys in the leaf nodes and keeps only the reference values of the leaf nodes in a Judy structure. Thus, J⁺-tree outperforms Judy in range queries and sequential scans for indexed nested looping joins. Our experimental study confirms this point and in the meantime shows that J⁺-tree has better performance than Judy in insertion operation. We also compare J⁺-tree with T-tree and B⁺-tree. The results demonstrate that J⁺-tree outperforms T-tree and B⁺-tree in terms of search and update time.

The rest of this paper is organized as follows. Section 2 is the related work. We present our J⁺-tree index in Section 3. The theoretical analysis of various index structures is in Section 4. Section 5 shows our detailed experimental comparisons. Finally, we conclude this paper and give the future work.

2 Related Work

In this section, we review several data structures: B⁺-tree, T-tree and Judy. All these structures belong to the same type - the order-preserving class. In fact, Hash is also considered as an index structure. But for the reason that it is not order-preserving, we exclude it in this paper.

2.1 B⁺-Tree

B⁺-tree [2] is a variant of B-tree and a popular index structure in traditional DBMS. In MMDB, B⁺-tree also gets much attention of many researchers. Some variants of B⁺-tree have been proposed, such as CSB⁺-tree [9] and pB⁺-tree [3] which optimize B⁺-tree in CPU cache behavior. But in general these new types are similar to B⁺-tree in the aspect that the tree depth increases with the number of keys.

2.2 T-Tree

T-tree [6], proposed by Lehman and Carey, is a type of binary tree with many elements in a node. The “T” in T-tree refers to the shape of the node data structure. Because T-tree is rooted by the AVL tree and the B-tree, it has the characteristics of these two structures. Firstly, it retains the binary search nature of the AVL tree, and its height is usually larger than that of B⁺-tree. Secondly, it keeps the storage feature of B-tree, so that range queries need tree traversals which are slower than scanning linked leaf nodes.

2.3 Judy

The Judy array [1] was proposed by Doug Baskins, which is a fast associative array that can store and search values using integer or string keys. It is built using digital trees with flexible adaptable nodes and various compression tricks. The size of Judy is not pre-allocated and is adjusted dynamically according to the number of keys, and compared with pure digital trees, Judy consumes much less memory by choosing data structures of every node appropriate for the population below the node. Judy takes the good usage of CPU cache into account and needs no external tuning parameters.

Judy has three kinds of branch nodes - linear branch, bitmap branch and uncompressed branch, and two kinds of leaf nodes - linear leaf and bitmap leaf. The choice of the node data structure depends on the population of the keys under the node - when a linear branch overflows it may become a bitmap branch. There are four types of Judy arrays in Judy family: Judy1, JudyL, JudySL and JudyHS. JudyL is the most important type which maps a long word to a one-word value and is the workhorse of JudySL and JudyHS.

Logically, Judy is a 256-way digital tree, and keys are decoded into bytes. Each level stores at least one byte of the key and portions of the key are kept in different nodes throughout the tree. Although Judy is not a height-balanced tree, the maximum height is predictable. It has at most four levels for 32-bit keys and eight levels for 64-bit keys. Thus, in contrary to B⁺-tree and T-tree, search time in Judy does not depend on the number of keys stored in the tree, which makes the search operation very fast. Update can also benefit from this advantage because search cost is part of update cost. It is this good characteristic that encourages us to consider Judy as the base structure of our new index in main memory databases. Judy has some drawbacks, which make it not sufficient

for a database index - firstly range queries are more difficult to implement and slower than in B⁺-tree, and secondly, Judy does not support duplicate keys and only one share of keys can be stored in the tree. Our J⁺-tree index can eliminate these limitations.

3 J⁺-Tree

In J⁺-tree, all the keys are placed in the double linked leaf nodes. A Judy structure, the upper part of J⁺-tree, is used to store the minimum key of each leaf node as the reference value towards the leaf node. This design makes J⁺-tree have the good structure advantage of Judy, and more suitable to be an index for main memory databases.

3.1 Definition

The definition of J⁺-tree index is as follows:

- (1) J⁺-tree is comprised of a Judy structure and double linked leaf nodes.
- (2) All the keys are stored in the leaf nodes in order. The copy of the minimum key of each leaf node is stored in the Judy structure as a reference value to the leaf node.
- (3) Judy structure consists of pairs of a reference value and a pointer. The pointer points to the leaf node whose keys are equal to or larger than the corresponding reference value.
- (4) A leaf node contains at most m entries and no fewer than $\lceil m/2 \rceil$ entries. Each entry has a key value and a pointer to the location of the record matching the key.
- (5) Each leaf node has two extra pointers pointing to the previous leaf node and the next leaf node.

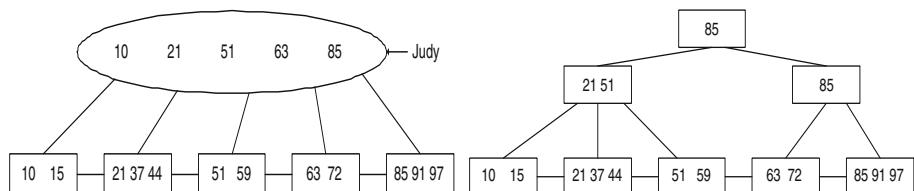


Fig. 1. J⁺-tree

Fig. 2. B⁺-tree

From the definition we can see that J⁺-tree is somewhat like B⁺-tree in the aspect of leaf nodes. But the part above leaf nodes is totally different. First, the storage structure is not the same. J⁺-tree uses a Judy tree to store the reference values while B⁺-tree uses an n -way tree. Second, the number of reference values is different. In J⁺-tree, the number of values in Judy is the same as the number of leaf nodes since only the minimum key of each leaf node is stored in Judy.

While in B^+ -tree, besides the minimum keys of leaf nodes, the minimum keys of nodes in lower levels are also kept in the nodes of higher levels. A simple example is shown in Figure 1 and Figure 2. Figure 1 is J^+ -tree ($m = 3$), and Figure 2 is the B^+ -tree.

3.2 Operations for J^+ -Tree

Search Algorithm. Searching a key in J^+ -tree is as follows:

- (1) First, look for the reference value in Judy according to the search key. If found, arrive at the leaf node via the corresponding pointer, otherwise, arrive at the first leaf node. This leaf node is called the bounding node that bounds the search key.
- (2) In the leaf node, examine whether the search key is present by a binary search. If the key is found, the search succeeds, otherwise, fails.

In step (1), how to look for the reference value in Judy is important. The algorithm works as follows:

- a) Look for the search key in Judy. The process of searching a key in Judy is somewhat like searching a digital tree. Each level in Judy stores at least one byte of the key. Decode the key into several bytes, and search the Judy tree according to the corresponding bytes. If the key is found, the algorithm succeeds and stops; otherwise, it proceeds to step b).
- b) Step a) stops at a certain Judy node. Start with the current node to look for the value less than the search key by backtracking through the nodes that were just traversed. If a value less than the search key is found, the algorithm succeeds; otherwise it fails.

Besides single value selection, there is another typical query operation in database applications - range query. For range query:

- (1) Use the minimum value of the range query as the search key and find the bounding node.
- (2) Begin from the bounding node to find all the keys within the range of query along the double linked leaf nodes.

Insertion Algorithm. Before the insertion of a key, search the J^+ -tree to find the bounding node. The new key is inserted into the bounding node. If the node overflows, split the leaf node into two nodes. The details are as follows:

- (1) Search the J^+ -tree for the bounding leaf node.
- (2) If the leaf node is not full, just insert the new key into it. If the new key is the minimum value of this node and the bounding node is the first leaf node, delete the reference value of this leaf node from Judy and insert the new reference value into Judy.
- (3) If the leaf node is full (contains m entries), split the leaf node into two leaf nodes (the original leaf node is the first node, and add a new leaf node to be

the second node). Insert the new key into the appropriate node. Each node has no fewer than $\lceil m/2 \rceil$ entries. Insert the minimum key of the second leaf node into Judy as its reference value. For the first leaf node, if necessary (as described in step (2)), update its reference value in Judy.

Deletion Algorithm. The search operation is the first step in the deletion operation. The algorithm is as follows:

- (1) Search for the bounding leaf node and examine whether the key to be deleted exists. If not, return failure.
- (2) If the bounding node contains more than $\lceil m/2 \rceil$ entries, just delete the key. Even if the deletion key is the minimum in the leaf node, the reference value in Judy need not be updated for the efficiency reason.
- (3) If the bounding node contains $\lceil m/2 \rceil$ entries, delete the key, and
 - a) If the previous node of the bounding node exists and contains more than $\lceil m/2 \rceil$ entries, move its maximum into the bounding node as the new minimum. Update the reference value of the bounding node in Judy. Else
 - b) If the next node of the bounding node exists and contains more than $\lceil m/2 \rceil$ entries, move its minimum into the bounding node to become the new maximum. Update the reference value of the next node in Judy. Else
 - c) If the previous node exists, merge the bounding node with its previous node. Delete the reference value of the bounding node from Judy and discard the bounding node. Else
 - d) If the next node exists, merge the bounding node with its next node. Delete the reference value of the next node from Judy and discard the next node.

3.3 Additional Illustration About J⁺-Tree

The leaf nodes of J⁺-tree are double linked so that the previous node which is needed in deletion operation can be quickly found. Duplicate keys can be supported easily in J⁺-tree - the duplicates can be stored in leaf nodes and the upper Judy structure still keeps only the distinct values. This causes some slight changes for the definition and operations of J⁺-tree. Due to limited space no more details are provided in this paper. If the keys are 4-byte long such as typical integers, J⁺-tree utilizes JudyL to store the reference values (64-bit systems are not considered in this paper). If the keys have a larger size, which is the same problem as [9] has encountered, our method is like that in [9] - put the keys in a *domain* and store the corresponding IDs in J⁺-tree.

4 Time and Space Analysis

In this section, we analyze the theoretical time and space performance of B⁺-tree, T-tree, Judy and J⁺-tree. To simplify the presentation, we assume that a key and a pointer take the same space K which is four bytes. We let n denote the

number of keys, m denote the maximum number of keys per node (“node” does not include the node in Judy and the internal node in J^+ -tree), and c denote the node size. For B^+ -tree, c is $(2K * m + 2K)$ where the second “ $2K$ ”, which also exists in the nodes of T-tree and the leaf nodes of J^+ -tree, is used by the pointer to the next node in leaf nodes or the pointer to the first child node in internal nodes or/and some extra information. In T-tree c is equal to $(2K * m + 2K + 4K)$, where “ $4K$ ” refers to the space occupied by a parent pointer, a left pointer, a right pointer and a balance factor. For the leaf nodes in J^+ -tree, c is calculated using $(2K * m + 2K + K)$ where the last item “ K ” is the space taken by the pointer to the previous node. Some extra explanation about T-tree is needed: we store the copies of keys and the pointers to the records in its nodes, not just the pointers, because if so search is much slower due to indirection [8].

Table 1. Time Analysis

Method	Branching factor	Number of levels	Total comparisons	Comparisons of per internal node	Comparisons of per leaf node
B^+ -tree	$m+1$	$O(\log_{m+1} n)$	$O(\log_2 n)$	$O(\log_2 m)$	$O(\log_2 m)$
T-tree	2	$O(\log_2^{n/m})$	$O(\log_2^n)$	$O(\log_2 m)$	$O(\log_2 m)$
Judy	256	4	$O(1)$	$O(1)$	$O(1)$
J^+ -tree	256	5	$O(\log_2^m)$	$O(1)$	$O(\log_2^m)$

Table 1 shows the branching factor, number of levels and number of key comparisons in search operation. B^+ -tree has a branching factor related to the parameter m , while the other three structures have relatively stable branching factors. The depth of Judy and J^+ -tree is fixed, and the maximum is four levels for Judy and five levels for J^+ -tree. But the number of levels in B^+ -tree and T-tree depends on the number of keys and the number of keys in one node. If the parameter m is fixed, the depth of B^+ -tree and T-tree will become larger with the increase of the number of keys. Because T-tree is a binary tree, it is usually deeper than B^+ -tree. In the total comparisons, B^+ -tree and T-tree have much more comparison times than Judy and J^+ -tree since m is much less than n . The comparison operation of Judy has a constant order, which is true of the comparison of internal nodes in J^+ -tree. Due to the comparisons of leaf nodes, J^+ -tree has an order of \log_2^m time complexity. As a whole, the tree depth and total comparisons of Judy and J^+ -tree have nothing to do with the parameter n , so the search cost is low and relatively constant. Search cost is an important part of update cost and [9] has claimed that the total insertion cost of B^+ -tree depends on the size of the tree, thus the update cost of J^+ -tree can also benefit from the low search cost.

Table 2 is the space consumed by various data structures in general. We assume all the nodes in B^+ -tree and the leaf nodes in J^+ -tree are 70% full, the internal space of B^+ -tree is acquired by multiplying $(1/m)$ by the leaf space [9] and the space per key in Judy is 8 bytes [12]. For T-tree, we assume that in

Table 2. Space Analysis

Method	Space
B ⁺ -tree	$(n/(0.7m)) * c * (1 + 1/m) = (n/(0.7m)) * (2K*m + 2K) * (1 + 1/m)$
T-tree	$(n/(2(m-1)) + n/m) * c = (n/(2(m-1)) + n/m) * (2K*m + 2K + 4K)$
Judy	$8n$
J ⁺ -tree	$(n/(0.7m)) * c + 8*(n/(0.7m)) = (n/(0.7m)) * (2K*m + 2K + K) + 8*(n/(0.7m))$

general the number of keys in one node is the minimum plus half of the difference between the maximum and the minimum.

5 Experimental Results

In this section we show the experimental comparisons of B⁺-tree, T-tree, Judy and J⁺-tree in time and space aspects in detail.

5.1 Experimental Setup

Our experiments were performed on a personal computer with 512M memory and 2.4GHz Intel Pentium 4 CPU, running Windows XP operating system. Our code was implemented in C language and was compiled by gcc in Cygwin environment.

We chose keys to be four-byte integers and tested various operations by using two kinds of keys, one is sequential and the other is random. These two types of keys are generated in advance and stored in arrays. Our test includes several parts listed below. In each part the number of keys that every index structure contains ranges from 500,000 to 11,477,960. When the tree is constructed with sequential keys, the keys used by other operations such as search, insertion and deletion are also sequential. This is also true of random keys.

Build - We inserted keys into an empty data structure. The inserts were each separate and only distinct values were allowed. Here we measured the build time (in fact the insertion cost) and memory space.

Search - To measure the search cost, we tested three operations for each index, single value selections for 1 million keys when the keys were in the tree, single value selections for 1 million keys when the keys were not present in the tree and 10,000 times of range queries where the difference between the lower and upper bounds was 3,000.

Insert - We inserted 0.5 million keys into each existing tree to measure the insertion cost. The inserted keys were picked from the data arrays prepared in advance.

Delete - To get the deletion cost we deleted 0.5 million keys from each existing tree. The deleted keys were those that had been stored in the indexes.

5.2 Implementation Details

We set the maximum number of keys in one node of B^+ -tree, T-tree and in each leaf node of J^+ -tree to 15. The key size and the pointer size are both 4 bytes. Thus the node size of B^+ -tree is 128 bytes, and that of T-tree is 144 bytes. The size of leaf node in J^+ -tree is 132 bytes. For T-tree, we set the minimum internal node size at two less than the maximum node size as in the original paper. For B^+ -tree, T-tree and the leaf nodes of J^+ -tree, a large memory pool is allocated in advance, so in the runtime, their memory space needed is acquired from the pool and the unused space is not freed directly but linked back to the pool. For Judy, we implemented its range query using the function of acquiring next key. In the implementation of our J^+ -tree, we used the value areas of Judy to store the pointers to the leaf nodes.

5.3 Results

We tested the build, search, insertion and deletion operations for sequential keys and random keys, respectively. In the figures, “seq” represents that keys are sequential and “rand” means keys are random.

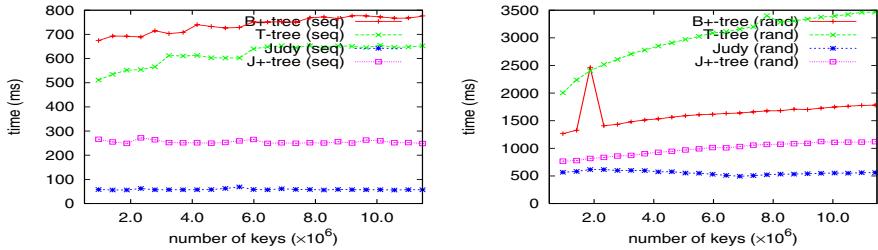


Fig. 3. Hit Search Time

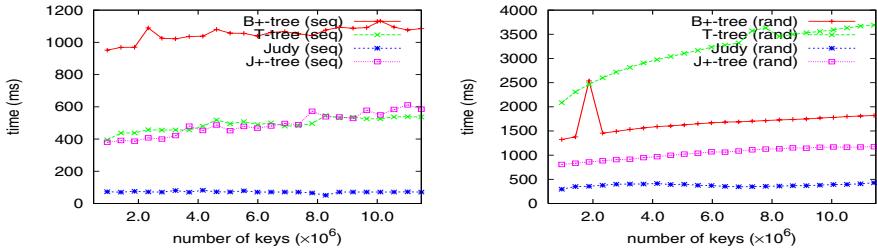


Fig. 4. Miss Search Time

Figure 3 shows the time of searching 1M keys when all the selected keys are present in the trees. For both sequential keys and random keys, J^+ -tree and Judy perform much better than B^+ -tree and T-tree, and their performance doesn't show obvious degrade as the tree sizes get larger, especially for sequential keys.

It is because, when the number of keys becomes larger, the depth of B⁺-tree and T-tree increases. When the keys are sequential all the indexes spend less time than that of random keys. Figure 4 shows the search time when the selected keys are not in the indexes. Again, for random keys the J⁺-tree and Judy indexes are better than the other two indexes. But for sequential keys J⁺-tree is similar to T-tree. This is because of the particularity of the search keys - all the keys we chose are larger than the maximum key in the indexes, so the search of tree can easily get to the next level via a few comparisons in T-tree.

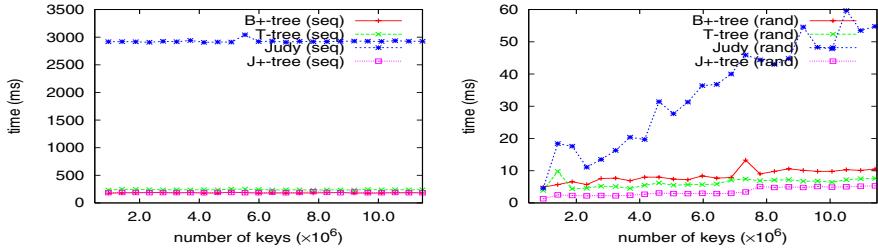


Fig. 5. Range Query Time

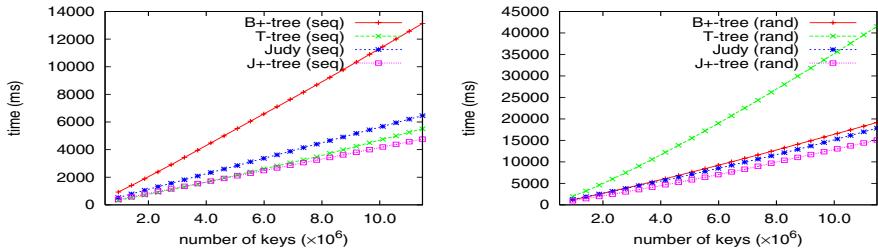


Fig. 6. Build Time

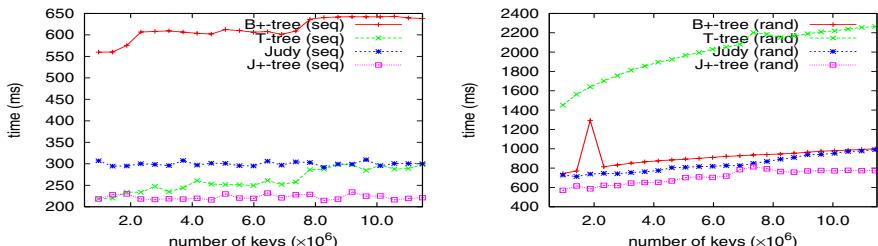


Fig. 7. Insert Time

Figure 5 is the result of range queries. We can see that the behavior of Judy is the worst and J⁺-tree is the best. For sequential keys, B⁺-tree performs as well as J⁺-tree and T-tree performs worse than them. When the keys are random, there are only a few keys in the range of query and just one or two leaf nodes of

B^+ -tree are scanned, thus the advantage of B^+ -tree is not exerted fully, which makes B^+ -tree slightly worse than T-tree.

The time to build each tree is shown in Figure 6. Figure 7 is the time of inserting 0.5M keys when the tree size is varied from 500,000 keys to 11,477,960 keys. In fact, they both show the insertion cost of various indexes, so the relative performance is the same. J^+ -tree is better than all the other indexes no matter what type the keys belong to. There are several reasons. First, J^+ -tree spends less time to find the right location to insert the key. Second, the Judy structure in J^+ -tree contains fewer keys than the Judy index so that there are fewer internal node transformations caused by insertions. Third, there are no cascade splits in J^+ -tree and if one leaf node is full, it is split into two leaf nodes, but this does not result in the split of the parent node as is in B^+ -tree.

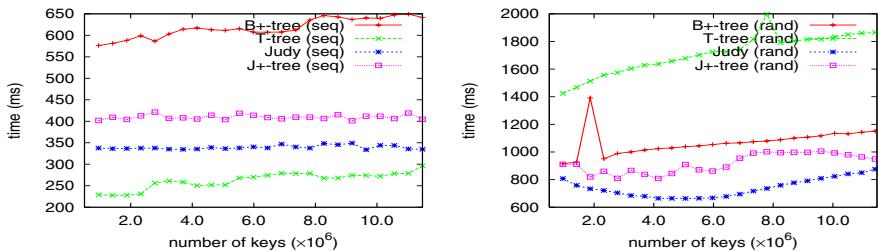


Fig. 8. Delete Time

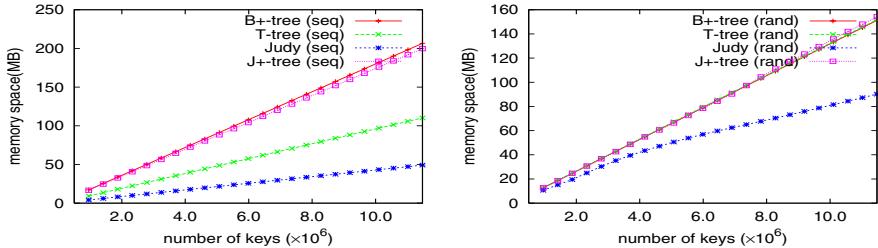


Fig. 9. Memory Space

Figure 8 shows the deletion time. For random keys, J^+ -tree is worse than Judy but still better than B^+ -tree and T-tree. Compared with Judy, J^+ -tree needs the node merge and key shift operations that consume much time. For sequential keys, J^+ -tree does not perform as well as T-tree. This is also due to the particularity of the deleted keys. Firstly, the key to be deleted each time is the minimum key in the trees, so T-tree can find the key quickly. Secondly, the deleted key is always in the leaf node, thus no too much extra work such as key shift is needed.

The space cost of various indexes is shown in Figure 9. Judy is indeed a memory efficient data structure. No matter what type the keys are it needs the lowest space. When the keys are sequential, the T-tree occupies less space than

J⁺-tree and B⁺-tree. It is because, for sequential keys the fill factor of the nodes in B⁺-tree and the leaf nodes of J⁺-tree is low - a little more than 50%, which results in much wasted space. When the keys are random the fill factor becomes larger, so for random keys, there is no distinct difference among J⁺-tree, B⁺-tree and T-tree.

6 Conclusions and Future Work

In this paper, we proposed a new index structure called J⁺-tree. It outperforms Judy in range queries, sequential scans and insertion operations, and is more suitable to be an index for main memory databases. Compared with B⁺-tree and T-tree, our analytical and experimental results show that J⁺-tree provides better performance in many aspects, due to an important reason from the good structure characteristic of J⁺-tree - the tree depth is fixed. In the future work, we will study the CPU cache behavior of J⁺-tree, and optimize our index by minimizing cache misses.

References

1. Baskins D.: Judy functions - C libraries for creating and accessing dynamic arrays. <http://judy.sourceforge.net>
2. Comer D.: The Ubiquitous B-Tree. ACM Computing Surveys. 11(2),(1979)
3. Chen S., Gibbons P. B., Mowry T. C.: Improving Index Performance through Prefetching. In Proceedings of the SIGMOD 2001 Conference. (2001)235-246
4. Garcia-Molina H., Salem K.: Main Memory Database Systems: An Overview. IEEE Transactions on Knowledge and Data Engineering. 4(6),(1992)
5. Jung K., Lee K.: Design and Implementation of Storage Manager in Main Memory Database System ALTIBASE
6. Lehman T. J., Carey M. J.: A Study of Index Structures for Main Memory Database Management Systems. In Proceedings of the 12th VLDB Conference. (1986) 294-303
7. Lehman T. J., Shekita E. J., Cabrera L.: An Evaluation of Starburst's Memory Resident Storage Component. IEEE Transactions on Knowledge and Data Engineering. 4(6), (1992)555-566
8. Rao J., Ross K. A.: Cache Conscious Indexing for Decision-Support in Main Memory. In Proceedings of the 25th VLDB Conference. (1999)
9. Rao J., Ross K. A.: Making B⁺-Trees Cache Conscious in Main Memory. In Proceedings of ACM SIGMOD Conference. (2000)
10. Rastogi R., Seshadri S., Bohannon P., Leinbaugh D., Silberschatz A., Sudarshan S.: Logical and Physical Versioning in Main Memory Databases. In Proceedings of the 23rd VLDB Conference. (1997)86-95
11. The TimesTen Team: High Performance and Scalability through Application-Tier, In-Memory Data Management. In Proceedings of the 26th VLDB Conference. (2000)
12. Programming with Judy: C Language Judy Version 4.0. <http://docs.hp.com/en/B6841-90001/index.html>

CST-Trees: Cache Sensitive T-Trees

Ig-hoon Lee¹, Junho Shim², Sang-goo Lee³, and Jonghoon Chun⁴

¹ Prompt Corp., Seoul, Korea
ihlee@prompt.co.kr

² Department of Computer Science, Sookmyung Women's University, Korea
jshim@sookmyung.ac.kr

³ School of Computer Science & Engineering, Seoul National University, Korea
sglee@europa.snu.ac.kr

⁴ Department of Computer Engineering, Myongji University, Korea
jchun@mju.ac.kr

Abstract. Researchers have modified existing index structures into ones optimized for CPU cache performance in main memory database environments. A Cache Sensitive B+-Tree is one of them. It is designed to minimize the impact of cache misses for B+-Trees and it has been known to be more effective than other types of main memory index structure including T-Trees. In this paper, we introduce a Cache Sensitive T-Tree (CST-Tree) and show how T-Trees can also be redesigned to be cache sensitive. We present an experimental performance study which shows that our Cache Sensitive T-Trees can outperform the original T-Trees and Cache Sensitive B+-Trees on average 65 percent and 17 percent, respectively.

1 Introduction

As random access memory becomes more condensed and cheaper, it becomes feasible to store and manage database within large main memories. Researchers have paid attention to various aspects of main memory databases. The index structure for main memory is one area in which T-Trees were proposed as a prominent index structure for main memory [6].

Recently, [8] and [9] claimed that B-Trees may outperform T-Trees owing to the current speed gap between cache access and main memory access. CPU clock speeds have been increasing at a much faster rate than memory speeds [1, 3, 7]. The overall computation time becomes more dependent on first level instruction cache misses (L1) and second level data cache misses (L2) than on disk buffer misses. The total number of memory accesses for T-Trees is higher than the one for B+-Trees, in that T-Trees are designed considering on random access and pointer operations [9]. In the past we considered the effect of buffer cache misses to develop an efficient disk-based index structure. The same applies to the effect of cache misses.

Albeit cache optimization in main memory systems in principle is similar to main memory optimization in a disk-based system, a significant difference is that the management of the cache is done by the hardware and the database system does not have a direct control to improve the *cache hit*, memory references satisfied by the cache.

This is why the database system needs a built-in cache optimized index structure. A careful design considering the characteristics of cache behavior and cache replacement policy may lead to improvement of cache hits. A most well-known cache optimized index structure for main memory database systems is CSB+-Trees (Cache Sensitive B+-Trees) that is a variant of B+-Trees [9].

In this paper, we study how to design the existing T-Trees index structure to better utilize the cache and introduce a new index structure CST-Trees (Cache Sensitive T-Trees). We analyze the complexity of CST-Trees, and conduct the experiment to check its performance. The experimental result show that our new cache sensitive T-Trees may outperform the original T-Trees and other existing index structures: CSB+-Trees and B+-Trees.

The rest of this paper is structured as follows. Section 2 presents the related work. The original T-Trees and our analysis with regard to its cache consciousness are provided. In Section 3 we introduce our modified cache-conscious Trees and provide the basic algorithms. In Section 4 we present the experimental performance study. And finally, conclusions are drawn in Section 5.

2 Related Work

Most widely used tree-based index structures may include AVL-Trees, B+-Trees, and T-Trees [6]. The AVL-Tree is a most classical index structure that was designed for main memory [5]. It is a binary search tree in which each node consists of one key field, two (left and right) pointers, and one control field to hold the balance of its subtree (Figure 1-(a)). The left or right pointer points the left or right sub-trees of which nodes contain data smaller or larger than its parent node, respectively. The difference in height between the left and right sub-trees should be maintained smaller or equal to one. If an update affects a leaf node and leaves the tree unbalanced, i.e., a control field is larger than 11, a rotation operation is performed. There are four different rotation operations; LL (Figure 1-(b)), LR (Figure 1-(c)), RR, and RL. The RR and RL operations are symmetric to LL and LR, respectively.

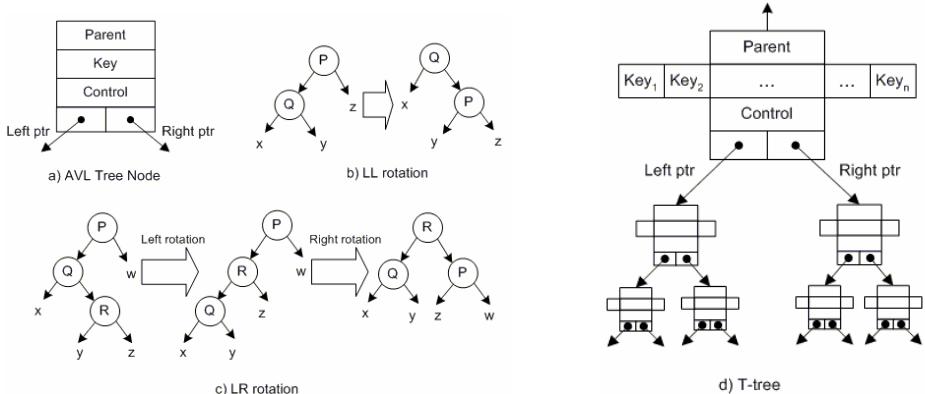


Fig. 1. AVL Trees and T-Trees

The major disadvantage of an AVL-Tree is its poor storage utilization. Each tree node holds only one key item, and therefore rotation operations are frequently performed to balance the tree. T-Trees address this problem [6]. In a T-Tree, a node may contain n keys (Figure 1-(d)). Key values of a node are maintained in order. Similar to an AVL-Tree, any key stored within a left and right sub-tree should be smaller or larger than the least and largest data of a node, respectively. The tree is kept balanced by the same rotation operations as for the AVL-Tree.

B-Trees [5] are designed for disk-based database systems and need few node accesses to search for a data since trees are broad and not deep, i.e., multiple keys are used to search within a node and a small number of nodes are searched. Most database systems employ B+-Trees, a variant of the B-Tree. In [8] and [9], authors showed that B+-Trees have a better cache behavior than T-Trees, and suggested to fit a node size in a cache line, so that a cache load satisfy multiple comparisons. They introduced a cache sensitive search tree [8], which avoids storing pointers by employing the directory in an array. Although the proposed tree shows less cache miss ratio, it has a limitation of allowing only batch updates and rebuilding the entire tree once in a while. They then introduced an index structure called CSB+-Tree (Cache-Sensitive B+-Tree) that support incremental updates and retain the good cache behavior of their previous tree index structure [9]. Similar to their previous tree structure, a CSB+-Tree employs an array to store the child nodes. However, it now has one pointer for the first child node and the location of other child nodes is calculated by an offset to the pointer value. We used a similar approach to reduce the pointers within a node.

3 Cache Sensitive T-Trees

3.1 Cache Insensitiveness of T-Trees

The reasons that T-Trees are not quite effective to utilize the cache compared to other index structures such as B+-Trees are as follows. First, cache misses are rather frequent in T-Trees. The height of a T-Tree is much higher than the one of a B+-Tree. That is, the total number of memory accesses from the root to the leaf node is higher in T-Trees. Another reason that a T-Tree has higher cache misses is due that it does not align the node size with the cache line size. As shown in [2, 4, 8, 9], setting the node size with the cache line size is indeed desirable to decrease the cache miss of an index structure.

Secondly, in T-Trees, much portion of data brought to the cache is not actually used. Whenever the processor wishes to access a memory location for a node and the location is not found in a cache, i.e., cache miss, a copy of data for the location is transferred from memory into cache. This transfer incurs a delay since main memory is much slower than cache. Within the copied data in cache T-Trees use only two keys (maximum and minimum keys) for comparison and access another memory location for another node. In contrast, B+-Trees use $\lceil \log_2 n \rceil$ keys that are brought to the cache for binary search comparison. Additionally, T-Trees use a record pointer for each key within a node, which leads the half of the node space is not utilized but wasted in the cache.

3.2 Cache Sensitive T-Trees

In this section we present the Cache Sensitive T-Trees and describe how we make the original T-Trees more cache-conscious by resolving cache-insensitiveness.

Higher usage of cached data: For T-Trees, the only data used for comparison within a node are its maximum and minimum keys. In a modified T-Tree [6], only maximum key is used for comparison. We construct a binary search tree which consists of only the maximum keys of each node (Figure 2-(b)). We use the binary search tree as a directory structure to locate a node that contains an actual key that we are looking for. The size of the binary search tree is not big and great portion of it may be cached. More importantly, the cached data will be hit high since every searching explores the tree first.

Removal of pointers: First, if a binary tree is represented as an array, there is no need to store explicit pointers to the child or parent nodes. If a node is stored at index i in an array and the root is at 1, then its parent, left and right child nodes may be found at $i/2$, $i*2$, and $i*2 + 1$, respectively. Secondly, when the child node groups of any given node group are stored contiguously, we need only one child pointer to indicate a first child node group explicitly (Figure 2-(c)).

Alignment of node size with cache line size: We make a binary search tree as full as possible given an array which size is the same to the cache line. We call each binary search tree in an array a *node group*. For example, given that keys are 4 bytes integers, if a cache line size is 32 bytes, then a binary search tree in a node group may contain upto 7 keys and its height is 3 (Figure 2-(c)). We always align the size of each node group with cache line size, so that there will be no cache miss when accessing data within a node group i.e., a child node to access is indexed $i*2$ or $i*2 + 1 < (\text{cache line size}/\text{pointer size})$. We use pointers to access from a node group to other node groups. Obviously, cache misses are unavoidable when accessing across the node groups.

Now we introduce our modified T-Tree, called Cache Sensitive T-Tree, as follows.

CST Trees: The CST-Tree is a k -way search tree which consists of node groups and data nodes (assume that a node group can have $k-1$ keys).

(P1) Data node contains keys and node group consists of maximal keys of each data nodes.

(P2) Each node group is a binary search tree represented in an array.

(P3) The tree is balanced, i.e., difference in height between the left and right subtree is less than 2, and a binary search tree of any node group is also balanced.

(P4) Sub-trees are also CST-Trees.

3.3 Operations on a CST-Tree

In this section, we consider search, insert, delete and balance operations on CST-Trees.

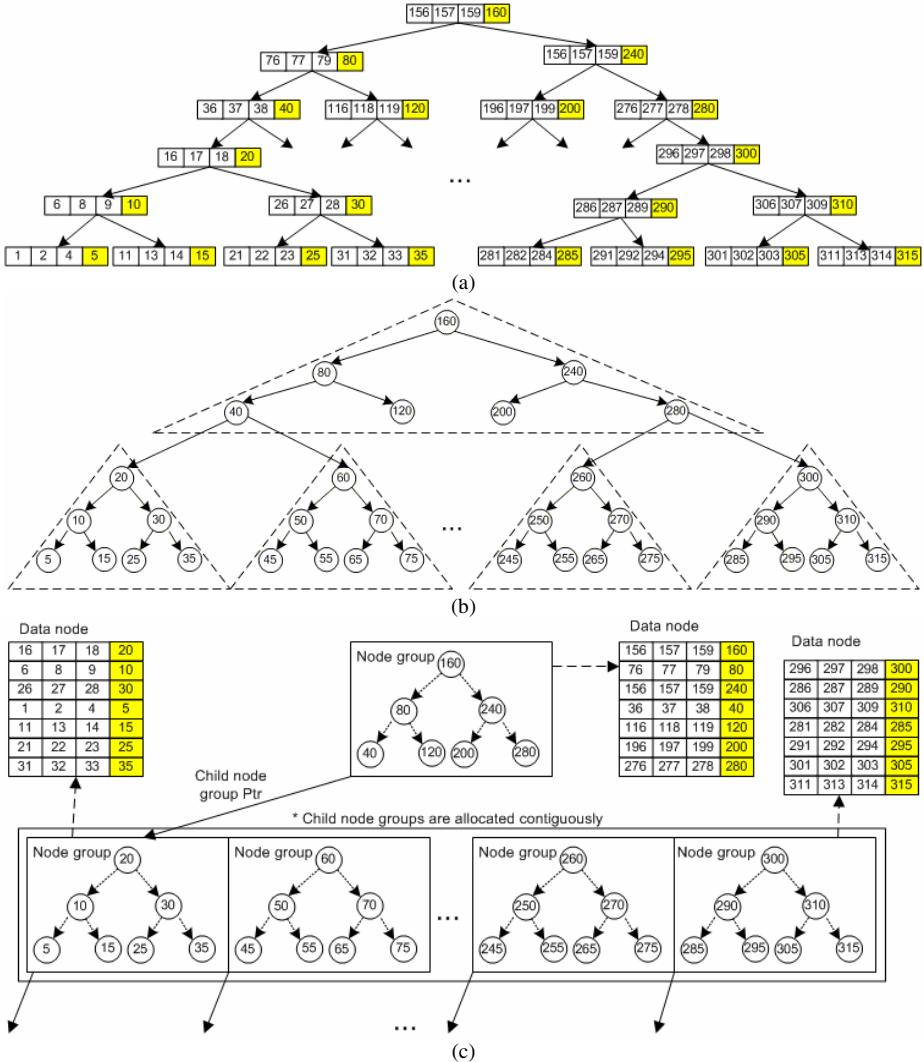


Fig. 2. (a) A T-Tree. (b) The Corresponding binary search tree with (a). (c) The corresponding CST-Tree with the (a).

3.3.1 Search Operation

Search algorithm of CST-Trees is different from T-Trees, since CST-Trees consist of node groups and data nodes. An illustrative example of a CST-Tree is shown in Figure 3.

First, accessing the root node group incurs 1 cache miss. Since the given key ‘287’ is bigger than the key ‘160’, ‘240’, and ‘280’, we access the second node group and it incurs second cache miss. In the second node group, since ‘287’ is smaller than ‘300’, we mark the current comparing position. ‘287’ is also smaller than ‘290’, therefore we move the mark to the current comparing position. In the leaf node group, after the last

key comparison, we do a binary search on the data node which corresponds to the last mark, and it incurs third cache miss. If there exists a given key at the data node, we have succeeded to find the search key. Otherwise we have failed.

During a search operation on CST-Trees from the root to the leaf node group, only accessing a sub-tree (child node group) and a data node incur cache misses. Doing a binary search in a node group does not incur a cache miss, because the size of a node group is the same as one of a cache line. Therefore the number of cache misses of a CST-Tree search operation is “*CST-Tree height + 1*” (3 cache misses in Figure 3). We present the evaluation results for the number of cache misses on a search operation in section 4.2 and describe the time complexity in section 3.4.

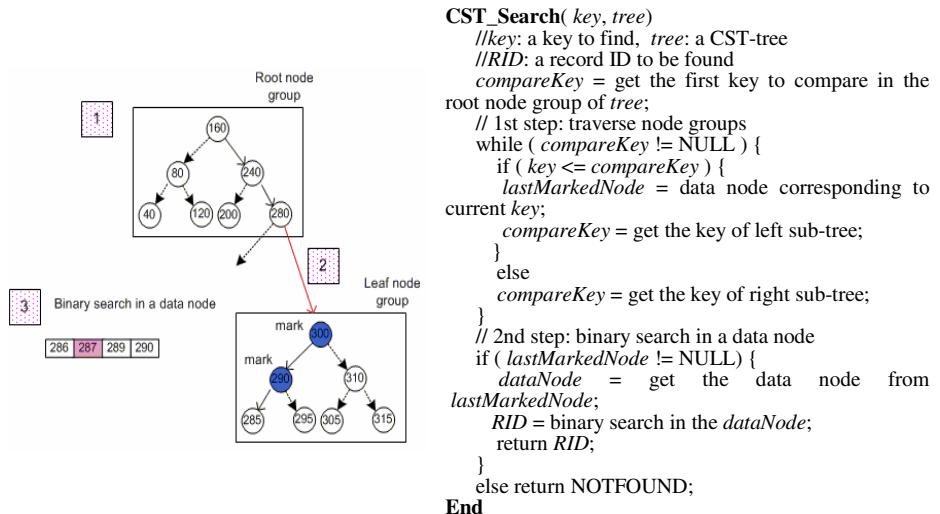


Fig. 3. An illustration of search operation in CST-Trees

3.3.2 Insert / Delete Operation

Insertion and deletion algorithms of CST-Trees are similar to T-Trees with an exception of a tree balancing algorithm (section 3.3.3).

An insertion operation is as follows. First, we find the data node to insert the given key and then insert the key to the corresponding data node. If the data node is not full, we simply insert the key to the data node. When the given key becomes a maximal key within the data node, we replace the key of the corresponding node group with the given key. If the data node is full, we delete the minimal key and insert the given key to the data node. Then we insert the deleted key into the left sub-tree as a maximal key. When there is no left sub-tree of the data node, we add a new data node (if we need to add a new node group, we have to add a new node group first) and insert the deleted key there.

As the balance check and rotation within the node group after the addition of a new data node is needed, balance check and rotation between node groups after the addition of a new node group is also needed. We present the rotation operation at section 3.3.3.

Figure 4 is an example of insertion operation of inserting the key ‘288’ into a CST-Tree. In Figure 4, the position to insert the new key is at the data node ‘A’ whose maximal key is ‘290’. Because the data node ‘A’ is full, we delete ‘286’ and insert the given key ‘288’ into ‘A’. When we insert the deleted key ‘286’ to the data node ‘B’, we delete ‘281’ again and insert ‘286’ into ‘B’. Since there is no left sub-tree, we add a new node group (because there is no room in the leaf node group, we need to add a new node group), add new data node, and insert ‘281’ into the new data node.

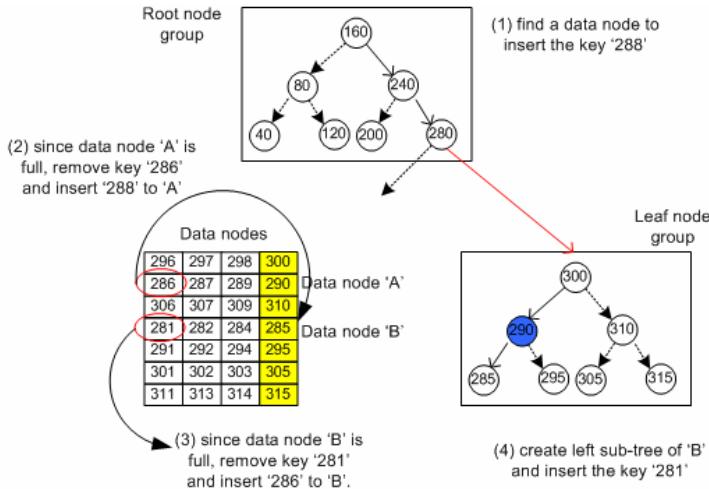


Fig. 4. An illustration of insertion operation in CST-Trees

We note here that deletion operation in CST-Trees is similar to the one in T-Trees except for a tree balance algorithm. Since it has analogy to opposite of insertion operation which is explained at the previous phrase, we do not describe the detail on deletion operation in CST-Trees.

3.3.3 Balance Operation

A CST-Tree is at whole a k-way search tree in which each node group contains binary search trees. For balancing the binary search trees we may apply a balancing algorithm similar to those of AVL Trees and T-Trees. A performance factor that we prioritize is the cache miss. Note that balancing a binary search tree does not cause a cache miss in that we align the node group size with the cache line size. However, every access to a non-cached node group causes a cache miss. Therefore we should pay more attention to balancing a CST-Tree across the node groups so as to minimize the average number of accesses to node groups.

Tree Balancing across Node groups

In this subsection, we explain how to balance a CST-Tree when a difference in height of its sub-trees goes beyond one. Albeit we present a detailed algorithm later, let us first see its basic operation, I-to-J rotation operation.

An I -to- $J(P, i, j)$ rotation is an operation to move the i th child node group of P to the j th child node group. For example, Figure 5 shows how a CST-Tree structure

changes after an I-to-J rotation operation on P when i and j are 3 and 2, respectively. The other case that i is less than j is symmetrical to this case, so it is not shown.

In Figure 5, Q is the 3rd child node group of P , b is the 2nd child node group of P . *separator* ('2') is the middle key of P between i th and j th child node group. We copy b to *tempPrevJthChild*, move *separator* '2' to Q (modified b) as the minimal key, and move all other keys of Q except for the maximal key to Q' . Then we move '5' of the maximal key of Q to the position of the previous *separator*, move *tempPrevJthChild* (b) to the 1st child node group of Q' , and move the 1st (x), 2nd (y), 3rd (z) child node group of Q to Q' as 2nd, 3rd, 4th child respectively. Finally, we move w to P as the 3rd (i th) child node group.

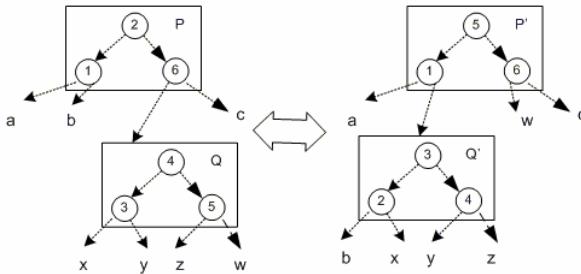


Fig. 5. A basic I-to-J rotation operation

```

CST_BalancingTree( $P$ : CST-Tree)
//  $P$  is a CST-Tree
if ( $P$  is an unbalanced CST-Tree) //  $\maxH(P) - \minH(P) \geq 2$  {
    if ( $\maxI(P) > \minI(P)$ ) { // rotation to left
         $Q = \maxI(P);$ 
        if ( $\maxI(P) - \minI(P) \neq 1$ ) { //  $\minI(P)$  is not next to  $\maxI(P)$ 
             $to = \maxI(P) - 1;$ 
             $from = \minI(P);$ 
            for ( $i = from$ ;  $i < to$ ;  $i++$ ) {
                i_to_j_Rotation( $P$ ,  $i$ ,  $i+1$ );
            }
        }
        if ( $\maxI(Q)$  is not the rightmost of  $Q$ ) {
            for ( $i = \maxI(Q)$ ;  $i <$  maximal # of child node groups;  $i++$ ) {
                i_to_j_Rotation( $Q$ ,  $i$ ,  $i+1$ );
            }
        }
        i_to_j_Rotation( $P$ ,  $\maxI(P)$ ,  $\minI(P)$ );
    }
    else { // rotation to right. omit because it is similar to the rotation to left
        ...
    }
}
if ( $P->\text{parent}$  is not NULL) {
    CST_BalancingTree( $P->\text{parent}$ );
}
End

```

Fig. 6. The CST-Trees balancing algorithm for node groups

A detailed node group balancing algorithm using the basic I-to-J rotation operation is illustrated in Figure 6. In the algorithm, $\minH(p)$ and $\maxH(p)$ mean the minimum

and maximum value among the heights of the sub-trees that are children of a given node group P . $\minI(p)$ and $\maxI(p)$ are the array index values for the sub-trees that result in $\minH(p)$ and $\maxH(p)$.

3.4 Time Complexity

In this section, we discuss the time complexities of search, insert, delete, and balance operations of CST-Trees. Let us say that n is the number of keys, s is the number of keys that a T-Tree contains within a node, and m is the number of keys that a node group of a CST-Tree contains or that a node of a B+-Tree contains.

If we store s keys into a data node, then the height of a m -way search tree to contain n keys should be at least $\log_m(n/s)$. Each node group contains a binary search tree of which height is $\log_2 m$. Search operation requires navigating a CST-Tree from the root node group to a leaf node group, and then again searches for a key within a data node. Then search operation requires $\log_m(n/s) \times \log_2(m)$ to locate a target data node, $\log_2 s$ to find a key in a data node. Therefore, the time complexity of the search operation becomes $O(\log_2 n)$.

Our insert operation of CST-Trees needs to locate a target data node to which a key is inserted. If the target data node is already full, then the minimum key should be removed from the tree and inserted back into the left subtree of the target data node. In the worst case, an insert operation requires $O(\log_2 n)$ to locate a target data node, $O(\log_2 s)$ to delete a key from the binary search tree, and $O(\log_2 n)$ to insert the key into the left subtree. Therefore, the time complexity of a insert operation becomes $O(\log_2 n) + O(\log_2 s) + O(\log_2 n) = O(\log_2 n)$.

Our delete operation also needs to locate a target data node where the key to be deleted is stored, and then it can delete the key from the target data node. Similar to the insert operation, it needs additional operations to avoid the underflow of the tree. Therefore, the time complexity of a delete operation becomes $O(\log_2 n)$.

Finally let us analyze the time overhead that a rotation operation requires for balancing a CST-Tree after performing a insert or delete operation. In CST-Trees, a binary search tree within a node group is an array structure. Therefore a rotation operation requires the memory copies of node groups that need to be relocated. A basic I-to-J rotation needs to move “child node groups + 2” number of node groups; i.e., a source node group (I) to be rotated, the child node groups of the source node group, and the target node group (J). For example, in Figure 5, Q is a source node group to be rotated. Then its child node groups (x, y, z, w) need to be moved by memory copies. In addition, b (target node group) also needs to be moved. Assuming that a node group is 16 bytes and the array size is 3, we need to copy $6 * 16 = 96$ bytes of data. Furthermore, if a cache line size is 64 bytes, then the array size of a node group becomes 15. And we need to copy $17 * 64 = 1152$ bytes of data. Compared to this, in the same environment, a node split operation of CSB+-Trees needs to copy on average $8 * 64 = 512$ bytes of data [9]. In short, the overhead for a CSB+-Tree node split operation is about the half of the one for a CST-Tree rotation operation.

4 Performance Evaluation

4.1 Experimental Environment

We performed an experimental comparison of the proposed CST-Tree. We implemented CST-Trees in C and the program was compiled and built by the Sun One Studio 8 Forte Developer 7. We ran our experiments on an Ultra Sparc III machine (1.2GHz, 4GB RAM) running SunOS 5.9. The Ultra machine has a $<8M, 64B>$ ($<\text{cache size}, \text{cache block (line) size}>$) L2 level cache. We used the Performance Analysis Tool [10], a tool provided by Sun Microsystems, to measure the number of cache misses. We only considered the L2 level cache misses as they did in [9].

For the performance comparison, we implemented all the methods including T-Trees, B+-Trees, and CSB+-Trees. For the implementation of CSB+-Trees and T-Trees, we referred to the original sources [6, 9] that are proposed by the original authors. All the methods are implemented to support search, insertion, and deletion. We implemented “lazy” deletion since it’s the one used by CSB+-Tress and is more practically used.

We assumed that keys are 4 bytes integers and each pointer takes 4 bytes. All keys are chosen randomly within the range from 1 to 10 million. The keys are generated in advance to prevent the key generating time from affecting the measurements. The node sizes of all the methods are chosen to 64 bytes, same to the cache block size of the Ultra Sparc machine, since choosing the cache block size to be the node size was shown close to optimal [8, 9]. We repeated each test three times and report the average measurements. Note that this set up environment is equal to the one in [9] to conduct fair experiments.

4.2 Results

Searching

In the first experiment, we compared the search performance of the methods. We varied the number of keys in the leaf nodes, and then measured the time and the number of cache miss that were taken by 200,000 searches. Each search key value was randomly chosen from the generated keys. Figure 7 shows the result. In Figure 7-(a), CST-Trees show the best in speed, and CSB+-Trees, B+-Trees, and T-Trees follow the next. On average, CST-Trees are 17%, 38%, and 65% faster than CSB+-Trees, B+-Trees, and T-Trees. In Figure 7-(b), CST-Trees show the least number of cache misses among the methods, while CSB+-Trees, B+-Trees, and T-Trees follow the next. The larger the number of searches, the wider the gap between CST-Trees and others.

Insertion and Deletion

In the next experiment, we tested the performance of insertion and deletion. Before testing, we first stabilized the index structure by bulkloading 1 million keys, as they did in [9]. Then we performed up to 20K operations of insertion and deletion and measure the performance. In Figure 8-(a), full CSB+-Tress [9] show the best in insertion, while B+-Trees, CST+-Trees and T-Tress show comparable performance in their insertions. Original CSB+-Trees show the worst.

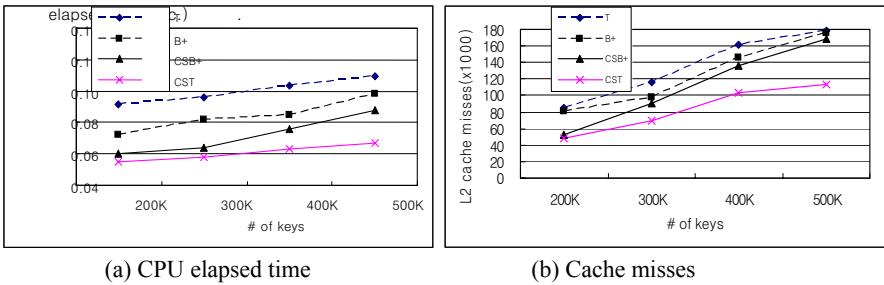


Fig. 7. 200K Searches after Bulkload

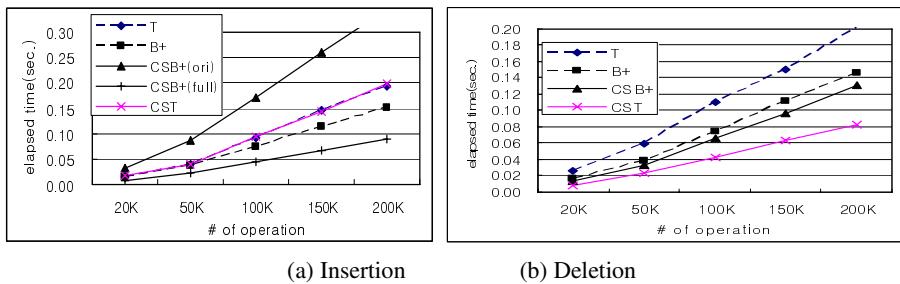


Fig. 8. CPU elapsed time for Insertion and Deletion operations

The delete performance shown in Figure 8-(b) follows a similar pattern to that of search. As mentioned earlier, we used “lazy” strategy for deletion. Most of the time on a deletion is spent on pinpointing the correct entry in the leaf node. Note that the actual elapsed time of each method for deletion takes a bit more time than for search in that we may need to go through several leaf nodes to locate the entry to be deleted.

5 Conclusion

In this paper, we proposed a new index structure called CST-Tree. CST-Trees are obtained by applying cache consciousness to T-Trees. Our analytical and experimental results show that CST-Trees provide much better performance than other existing main memory index structures owing to the better cache behavior. CST-Trees improve the search performance on average 17 %, 38%, 65% better than CSB+, B+-Trees, and T-Trees. CST-Trees also show comparable performance on insertion operations and better performance on deletion operations, although the performance benefits are less than in searching. As the gap between CPU and memory speed becomes widening, CST-Trees should be considered as a replacement of T-Trees in future.

Acknowledgements

We thank Mr. JaeYung Hur to his contribution at the initiation of this work. He worked with the authors to bring the idea of cache sensitiveness to T-Trees and helped much to implement the Trees. Without his contribution, this work would never been fully fledged. This work was supported in part by the Ministry of Information & Communications, Korea, under the Information Technology Research Center (ITRC) Support Program.

References

1. A. Ailamaki, et al., "DBMSs On A Modern Processor: Where Does Time Go?," in Proc. of the 25th Int'l Conf. on Very Large Database Systems, pp.266-277, 1999.
2. P. Bohannon, et al., "Main-Memory Index Structures with Fixed-Size Partial Keys," in Proc. of the 2001 ACM SIGMOD Int'l Conf. on Management of Data, pp.163-174, 2001.
3. P. Boncz, et al., "Database Architecture Optimized for the new Bottleneck: Memory Access," in Proc. of the 19th Int'l Conf. on Very Large Database Systems, pp.54-65, 1999.
4. T. M. Chilimbi, B. Davidson, and J. R. Larus, "Cache-Conscious Structure Definition," in Proc. of the ACM SIGPLAN 1999 conference on Programming language design and implementation, pp.13-24, 1999.
5. T. H. Cormen, et al., Introduction to Algorithms, The MIT Press, 1990.
6. T. J. Lehman, "A Study of Index Structures for Main Memory Database Management System," in Proc. of the 12th Int'l Conf. on Very Large Database Systems, pp.294-303, 1986.
7. S. Manegold, P. A. Boncz and M. L. Kersten, "Optimizing database architecture for the new bottleneck: memory access," VLDB Journal, Vol.9, No.3, pp231-246, 2000.
8. J. Rao, et al., "Cache Conscious Indexing for Decision-Support in Main Memory," in Proc. of the 19th Int'l Conf. on Very Large Database Systems, pp.78-89, 1999.
9. J. Rao, et al., "Making B+ Trees Cache Conscious in Main Memory," in Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data, pp.475-486, 2000.
10. Sun Microsystems, Inc., Sun ONE Studio 8: Program Performance Analysis Tools, available via "<http://docs.sun.com/app/docs/doc/817-0922>", 2003.

Specifying Access Control Policies on Data Streams

Barbara Carminati¹, Elena Ferrari¹, and Kian Lee Tan²

¹ DICOM, University of Insubria, Varese, Italy

{barbara.carminati,elena.ferrari}@uninsubria.it

² School of Computing, National University of Singapore, Singapore

tankl@comp.nus.edu.sg

Abstract. Many data stream processing systems are increasingly being used to support applications that handle sensitive information, such as credit card numbers and locations of soldiers in battleground [1236]. These data have to be protected from unauthorized accesses. However, existing access control models and mechanisms cannot be adequately adopted on data streams. In this paper, we propose a novel access control model for data streams based on the Aurora data model [2]. Our access control model is role-based and has the following components. Objects to be protected are essentially views (or rather queries) over data streams. We also define two types of privileges - Read privilege for operations such as Filter, Map, BSort, and a set of aggregate privileges for operations such as Min, Max, Count, Avg and Sum. The model also allows the specification of temporal constraints either to limit access to data during a given time bound or to constraint aggregate operations over the data within a specified time window. In the paper, we present the access control model and its formal semantics.

1 Introduction

In many applications, data arrive in the form of high speed data streams. Examples of such applications include telecommunication, battle field monitoring, network monitoring, financial monitoring, sensor networks, and so on. These data typically contain information that is sensitive and thus unauthorized accesses should be avoided. As an example, consider battle field monitoring, where the positions of soldiers are protected information that should only be accessible to the battleground commanders.

Clearly, there is a need to integrate access control mechanisms into data stream processing systems to achieve a controlled and selective access to data streams. However, to our knowledge, there has been no reported work that offers access control over data streams. From the data stream community, many data stream processing systems have been developed both academically (e.g., Aurora [2], Borealis [1], STREAM [3], TelegraphCQ [6]) and commercially (e.g., StreamBase [10]), but the focus in these systems has been on performance issues.

On the other hand, though the data security community has a very rich history in developing access control models [57], these models are largely tailored to traditional DBMSs. Thus, they cannot be readily adapted to data stream applications, mainly because: (a) traditional databases are static and bounded while data streams are unbounded and infinite; (b) queries in traditional DBMSs are one time and ad-hoc but queries over

data streams are typically continuous and long running; (c) in traditional DBMSs access control is enforced when users access the data; in data stream applications, access control enforcement is data-driven (i.e., whenever data arrive); (d) because of (c), access control is more computational intensive in data stream applications and specific techniques to handle it efficiently should be devised; (e) as data are streaming, temporal constraints (e.g., sliding windows) become more critical in data stream applications than in traditional DBMSs.

To cope with all these new requirements, in this paper, we propose a novel access control model for data stream applications based on the Aurora model [2]. We have decided to cast our research into the Aurora framework because (a) while there is still no consensus on a standard data model and query language for stream data, Aurora has emerged as one of the most relevant and mature proposals in the field, and (b) a full implementation of its processing engine is publicly available [1].

Our access control model is role-based and has the following components. Objects to be protected are essentially views (or rather queries) over data streams. As such, access can be granted only on selected tuples and/or attributes of a data stream, as well as only on selected attributes/tuples of joined streams. We also define two types of privileges - Read privilege for operations such as Filter, Map, BSort, and aggregate privileges for operations such as Min, Max, Count, Avg, and Sum. In addition, to deal with the intrinsic temporal dimension of data streams we introduce two temporal constraints - general constraints, that allow access to data during a given time bound, and window constraints, that support aggregate operations over the data within a specified time window. This last feature is very relevant for data streams since we can customize aggregate operations according to data sensitivity, by regulating the size of the window over which the aggregate function can be computed.

In this paper, we present the access control model by illustrating its syntax and formal semantics. We believe that the definition of an access control model on a formal basis is a key step to devise efficient methods for access control policy enforcement and to guarantee their correctness. To the best of our knowledge, this is the first reported work that proposes an access control model for data streams.

The remainder of this paper is organized as follows. In the next section, we provide some background to this work and the motivating scenario we will use throughout the paper. In Section 3 we present the proposed access control model, whereas Section 4 presents its formal semantics. Finally, we conclude this paper with directions for future work in Section 5.

2 Background

In this section, we first discuss a data stream scenario by highlighting its access control requirements. Then, we present a brief overview of the Aurora stream-processing engine.

2.1 A Motivating Scenario

In this section, we introduce an illustrative scenario that highlights some essential requirements of an access control model for stream data, in terms of policies it should be

able to express. Before doing that we briefly introduce how data streams are modeled throughout the paper. A stream consists of an append-only sequence of tuples with the same schema. In addition to standard attributes A_1, \dots, A_n the stream schema contains a further attribute, denoted as TS . TS stores the time of origin of the corresponding tuple, thus it can be exploited to monitor attributes values over time.

As a scenario, we consider the military domain presented in [2], where stream data are used to monitor positions and health conditions (e.g., heart beats, blood pressure) of platoon's soldiers. Positions and health conditions are modeled by means of two distinct streams, namely, *Position* and *Health*, with the following schemas: *Position* ($TS, SID, Pos, Platoon$) and *Health* ($TS, SID, Platoon, Heart, BPressure$), where the *SID* and *Platoon* attributes store soldier's and platoon's identifiers, respectively, both in the *Position* and *Health* streams, the *Pos* attribute contains the soldier position, the *Heart* attribute stores the heart beats, whereas the *BPressure* attribute contains the soldier's blood pressure value. We assume that users posing queries are identified by their roles, e.g., captains, soldiers, doctors, etc.

Let us now discuss some access control requirements that can arise in this scenario. Consider, for instance, the *Position* stream. Since the *Pos* attribute, modeling the position of a soldier, conveys sensitive information, it should be accessible only to selected users, such as for instance the captain of the soldier's platoon. In order to specify this requirement, *the access control model should support policies specified at the attribute level, in addition to the whole stream*.

Another important requirement is to be able to specify *policies that apply only on selected tuples within a data stream, identified on the basis of their content*. Consider, for instance, the case where we would like to grant captains access to the positions of soldiers not belonging to their platoons, only if they cross some specified borders. This requirement can be modeled as a policy granting the access to the *Pos* attribute only if the corresponding tuple satisfies two conditions, i.e., the condition stating that the corresponding soldier does not belong to the captain's platoon and the condition stating that the soldier has crossed a given border.

A further requirement is related to the temporal dimension of both data streams and some Aurora operators. For instance, sometimes it can be useful to constraint access to a data stream to selected temporal intervals. This is a relevant requirement, since *a data stream contains an intrinsic notion of time that should be exploited in the specification of temporal constraints*. For instance one can state that only during the action time doctors are authorized to monitor the heart beats of the soldiers.

Other examples of access control requirements related to temporal constraints are those authorizing window-based operators only on selected time windows. For instance, a doctor can be authorized to monitor the average of soldier's heart beats with arbitrary window size, if soldier belongs to the same platoon of the doctor, otherwise with a window with a maximum size of 1 hour. This limitation prevents doctors to infer whether and when there have been some critical situations (i.e., those characterized by heart beats with high frequency for a time longer than one hour). Thus, *an access control model for data streams should be able to support limitations on the windows over which window-based privileges can be exercised*.

In Section 3 we present an access control model to cope with all such requirements.

2.2 Aurora

In recent years much effort has been spent on the area of stream-processing engines [1236]. Among these engines, one of the most relevant and mature proposals is Aurora [2]. Aurora has been recently transferred to the commercial domain (i.e., the Stream-Base engine [10]), and redesigned with distributed functionalities (i.e., Borealis [1]). However, due to the scope of the paper in what follows we focus on Aurora stream-processing engine, by briefly introducing the underlying query model and algebra, and referring the interested readers to [2] for a detailed discussion of the core Aurora engine.

Aurora query model. In order to cope with the latency requirements implied by streams, Aurora exploits an ‘inbound processing’ instead of the traditional ‘outbound processing’, typical of conventional DBMSs, where data are stored and indexed before being queried. This means that query processing is performed directly on incoming streams. Aurora query processing exploits a dataflow paradigm, by modeling queries as a loop-free direct graph of operations (called boxes in Aurora), where tuples flow through all processing operations defined in the graph (called network in Aurora).

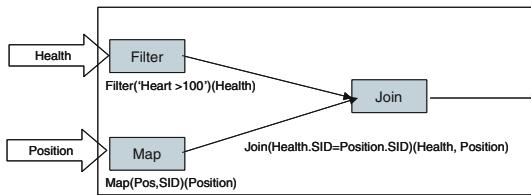


Fig. 1. An example of query in Aurora

An example of Aurora network is given in Figure 1, where two input streams, namely `Health` and `Position`, go through two different boxes (i.e., `Filter` and `Map`) before being joined together.

Aurora algebra. We provide now a brief overview of Aurora operators. In this paper, we focus more on the operators’ semantics rather than on details regarding their implementation. For this reason, for some of the Aurora operators described in [2] we provide a simplified syntax.

The first operator is the `Filter` box, which acts like a relational selection but having the capability to apply several distinct selections on a stream at the same time, and to route output tuples on the basis of satisfied predicates. The `Filter` syntax is $\text{Filter}(P_1, \dots, P_n)(S)$, where P_1, \dots, P_n are predicates over stream S . The result of the `Filter` operator consists of $n + 1$ different streams S^1, \dots, S^n , such that each stream S^j , contains those tuples of S satisfying predicate P_j , $j \in \{1, \dots, n\}$. Moreover, tuples that do not satisfy any predicate among P_1, \dots, P_n are returned in an additional stream S^{n+1} . Note that in the paper we do not consider the $(n + 1)$ th stream, i.e., the stream containing tuples not satisfying any predicate, by simply assuming that these tuples are not given output. Moreover, we assume that no `Filter` operation is performed if the set of predicates is empty. Let us consider the `Position`

stream of our motivating scenario, if we are interested in selecting only those soldiers whose position crossed a given border k , we can specify the following expression: $\text{Filter}(\text{Pos}>k)(\text{Position})$.

Another relevant operator of Aurora algebra is the **Map** box, which can be considered as a generalized projection operator. The syntax of **Map** is the following: $\text{Map}(A_i=F_i, \dots, A_j=F_j)(S)$, where $\{A_i, \dots, A_j\}$ is a subset of the attributes of S ' schema, and F_i, \dots, F_j are arbitrary functions over the input stream. Thus, instead of projecting the value of an attribute A_i , the **Map** operator projects the result of an arbitrary functions applied on it. In the paper, for simplicity, we consider a **Map** operator projecting only attributes' values. This can be obtained by applying only the identity function rather than arbitrary function, which is omitted in the following examples. Let us consider for instance the **Health** stream. $\text{Map}(\text{SID}, \text{Heart})(\text{Health})$ projects attributes **SID** and **Heart**. Moreover, if the set of attributes specified in the **Map** operator is empty, we assume that the operation returns the input stream.

In the Aurora algebra there is a further operator, **Bsort**, which sorts the tuples of a stream applying a bound pass bubble sort. The simplified syntax of this operator is the following: $\text{Bsort}(S)$. For instance, $\text{Bsort}(\text{Position})$ performs the bubble sort on the **Position** stream.¹

The Aurora algebra provides also an aggregate operator, i.e., **Aggregate** box, by which it is possible to apply both SQL-style aggregate operations and Postgres-style user-defined functions over data streams. Aggregate operators are evaluated according to a sliding window-based approach. This implies that the **Aggregate** operator receives as input both the size of the window and an integer specifying how to advance the window when it slides. The simplified syntax used throughout the paper is the following: $\text{Aggregate}(F, \text{Size } s, \text{Advance } i)(S)$, where F can be either an SQL-style aggregate operation or a Postgres-style user-defined function. As an example, $\text{Aggregate}(\text{Avg}(\text{Heart}), 2, 1)(\text{Health})$ returns the average of the soldier heart beats computed over windows with size 2 hours and advance step of 1.

The Aurora algebra also provides the **Join** operator. The join operator is a binary operator where the join predicate is specified as input. Throughout the paper, we adopt a variation that allows an arbitrary number of operators (not necessarily two). More precisely, we use the following syntax: $\text{Join}(P)(S_1, \dots, S_n)$ ². Let us consider once again, the **Position** and **Health** streams. $\text{Join}(\text{Position.SID}=\text{Health.SID})(\text{Position}, \text{Health})$ performs the natural join of **Position** and **Health** streams. Moreover, if no predicate is specified in the join operator, the result is the Cartesian product of the input streams.

A further operator is the **Resample** box, which can be helpful to align pairs of streams. The simplified syntax used in the paper is the following: $\text{Resample}(F)(S_1, \dots, S_n)$, where F is the interpolator function used in the semijoin-like synchronization. Finally, a further relevant operator is the **Union** box, which is used to merge a set of streams, having a common schema, into a unique output stream.

¹ According to the algebra in [2], it is possible to specify as input the assumed ordering over output streams.

² Note that the syntax proposed in [2], allows one to specify as input also the assumed ordering over input streams S_1, S_2 .

3 An Access Control Model for Data Streams

Generally, an access control policy states which subjects can access which (portions of) objects and under which conditions. Thus in designing an access control model for data streams we first need to specify the basic components of an access control policy, that is, the subject, protection object and privilege specification. Then, we will formally introduce access control policies for data streams, whereas their formal semantics is presented in Section 4.

Subject specification. We specify subjects according to a role-based approach [8]. Thus, access control policies associate permissions with roles, instead of with subjects, and subjects acquire permissions through their membership to roles. Examples of roles for our reference scenario are `soldier`, `doctor`, and `captain`.

Protection object specification. As pointed out in Section 2.1, an access control model for data streams should allow attribute-level and tuple-level access control. Thus, there is the need of an object specification flexible enough to represent, for instance, a whole stream, only selected stream's attributes, as well as only tuples satisfying certain conditions. To model such a variety of granularity levels, we borrow some ideas from how access control is enforced in traditional RDBMSs, where different granularity levels are supported through views. The idea is quite simple: define a view satisfying the access control restrictions and grant the access on views instead of on base relations. A view is defined by means of a CREATE VIEW statement, where the SELECT clause of the query defining the view specifies the authorized attributes, the FROM clause specifies a list of relation(s)/views, and the WHERE clause states conditions on attributes' values. We adopt the same idea to specify protection objects to which an access control policy applies. However, since a standard query language for data streams has not yet emerged³ we give a language independent representation of protection objects. Basically, we model a protection object by means of three components, which correspond to the SELECT, FROM and WHERE clauses of an SQL statement. Before presenting the formal definition of protection object specification we need to introduce some notations. In what follows, we denote with *Source* the set of all the streams to be protected, whereas given a stream S we denote with $S.\text{schema}$ the set of attributes in S 's schema.

Definition 1. (Protection Object Specification). A protection object specification p_obj is a triple $(\text{STRs}, \text{ATTs}, \text{EXPs})$, where:

- STRs is a set of streams $\{S_1, \dots, S_n\}$, belonging to *Source*;
- ATTs denotes a set of attributes A_1, \dots, A_l , where $A_j, j \in \{1, \dots, l\}$, belongs to the schema of the stream resulting from the Cartesian product $\{S_1 \times \dots \times S_n\}$ of the streams in STRs . If ATTs is equal to symbol ‘*’, it denotes all the attributes belonging to the schema of the stream resulting from the Cartesian product $\{S_1 \times \dots \times S_n\}$.
- EXPs is a boolean formula, built over predicates of the form: $A_i \oplus \text{value}$ or $A_i \oplus A_j$, where A_i, A_j are attributes belonging to the schema of the Cartesian product $\{S_1 \times \dots \times S_n\}$, \oplus is an operator of the Aurora algebra, and value is a value compatible

³ However, several research groups are doing a lot of work in this direction (see, for instance, [49]).

Table 1. Examples of protection object specifications

Streams	Attributes	Expressions
Position	*	-
Position	*	Position.Platon=X123
Health, Position	Health.Heart, Health.SID	Position.SID= Health.SID AND Position.Platon=X123

with the domain of A_i . If EXPs is omitted, it denotes all the tuples in the Cartesian product $\{S_1 \times \dots \times S_n\}$.

Given a protection object specification p_obj , we use the dot notation to refer to its components. According to Definition 1, a protection object specification p_obj identifies a view over the Cartesian product $\{S_1 \times \dots \times S_n\}$, where $S_j \in p_obj.\text{STRS}$, $j \in \{1, \dots, n\}$. The view is obtained by selecting from the Cartesian product all attributes specified in $p_obj.\text{ATTS}$, and by considering only those tuples satisfying conditions expressed in $p_obj.\text{EXPs}$.

Example 1. Table 1 presents three examples of protection object specifications defined according to Definition 1. The first protection object specification identifies the whole Position stream. By contrast, in the second protection object specification a condition on its content (i.e., $\text{Position.Platon}=X123$) is specified, thus identifying only those tuples of the Position stream having the Platoon attribute equal to X123. Finally, the third specification defines a protection object generated over the Cartesian product of Position and Health, where only Heart and SID attributes are projected. The condition expressed by the EXPs component ensures that only tuples having $\text{Position.SID}=\text{Health.SID}$ (i.e., join predicate) and referring to soldiers belonging to platoon X123 are considered.

Privileges. Privileges supported by the proposed access control model authorize all operations provided by Aurora (see Section 2.2), namely, Filter, Map, BSort, Union, Aggregate, Join and Resample. Instead of defining a different privilege for each operation, we assume that there exists a Read privilege which authorizes a subject to exercise the Filter, Map and BSort operations on a protection object, that is, all operations that require to read tuples from the data stream. Note that, if the Security Administrator (SA) wants to limit the right to read a stream only to selected attributes and/or tuples, the SA can grant the Read privilege directly on a protection object modeling the corresponding view. The same happens if the Read privilege has to be granted on the result of a join operation (or on a subset of its tuples). In contrast, our policy format does allow the SA to specify the Read privilege for a view consisting of the union of more data streams because we assume that the Union operation is authorized if the requesting user has the Read privilege on the two operand streams. The other class of privileges supported by our model, called Aggregate privileges, corresponds to aggregate functions allowed by Aurora. As introduced in Section 2.2, Aurora supports both SQL-style aggregate operations and Postgres-style user-defined functions. Here, to be as system independent as possible, we consider as aggregate functions only the standard SQL-style functions. Thus, the aggregate privileges are: Min, Max, Count, Avg, and Sum.

Temporal constraint specification. As discussed in Section 2.1, an access control model for data streams should be able to handle two different kinds of temporal constraints, that we call general and window-based constraints. Constraints of the first kind state limitations on the time during which subjects can exercise privileges on protection objects. They can be expressed in the form: $[\text{begin}, \text{end}]$, where begin and end are the lower and upper bounds, $\text{begin} \leq \text{end}$, and end can assume the infinite value.⁴ The begin and end values can be explicitly specified by the SA or can be returned by a predefined set of system functions \mathcal{SF} . For instance, we can assume a function $\text{TAction_start}(a)$, which returns the time TS when a given action a starts, and a function $\text{TAction_end}(a)$ that returns the time TE when an action a ends. It is important to note that by definition a stream always contains a temporal information, i.e., the timestamp TS . Therefore a general time constraint GTC identifies all and only those tuples satisfying the predicate: $TS \geq \text{begin} \wedge TS \leq \text{end}$.

For the second type of constraints, that is, those related to window-based operators, according to the Aurora algebra, a window is specified by two information: the window's size and the advance step. A window time constraint can therefore be defined by a pair: $[\text{size}, \text{step}]$, denoting the maximum size and advance step allowed in a window-based operation. Thus, by specifying a window time constraint in an access control policy granting an aggregate privilege p on a protection object σ , the SA is able to limit the window size and step according to which the aggregate operation can be executed. For instance, if the SA wants to limit a doctor to compute the average of heart beats of soldiers not belonging to his/her platoon only on a window with size 5 hours and a step of 2, he/she can state an access control policy granting the Avg privilege on the corresponding protection object by specifying $[5h, 2]$ as window time constraint.

We are now ready to formally define access control policies for data streams.

Definition 2. (Access control policy for data streams). An access control policy for data streams is a tuple: $(\text{sbj}, \text{obj}, \text{priv}, \text{GTC}, \text{WTC})$, where: sbj is a role; obj is a protection object specification defined according to Definition 1; $\text{priv} \in \{\text{Read}, \text{Min}, \text{Max}, \text{Count}, \text{Avg}, \text{Sum}\}$; GTC is a general time constraint; and WTC is a window time constraint.

Given an access control policy acp we denote with $\text{acp}.\text{sbj}$, $\text{acp}.\text{obj}$, $\text{acp}.\text{priv}$, $\text{acp}.\text{GTC}$ and $\text{acp}.\text{WTC}$ the sbj , obj , priv , GTC , and WTC components, respectively. We assume that all the specified access control policies are stored into a unique authorization catalog, called *SysAuth*. More precisely, *SysAuth* contains a different tuple for each access control policy, whose attributes store the access control policy components, as illustrated by the following example.

Example 2. Table 2 presents an example of *SysAuth* catalog, containing a set of access control policies for the Position and Health streams. The first policy authorizes captains and doctors to exercise the Read privilege (i.e., Filter, Map and Bsort operations) on the position and id of all and only soldiers belonging to their platoons, where this condition is modeled as a predicate (i.e., $\text{Position}.\text{Platoon}=\text{self}$).

⁴ We assume that begin and end value are specified by means of an SQL-like syntax.

Table 2. Examples of access control policies for data streams

Subject	Protection Object			Privilege	GTC	WTC	
	Streams	Attributes	Expressions			Size	step
Captain, Doctor	Position	Pos, SID	Position.Platon= self.Platon	Read	-	-	-
Captain	Position	Pos, SID	Pos \geq k	Avg	[TAction_start(a), TAction_end(a)]	1	1
Doctor	Health	Heart, SID	Health.Platon= self.Platon	Read	-	-	-
Doctor	Health	Heart, SID	Health.Platon \neq self.Platon	Avg	-	1	1
Doctor	Health, Position	Heart, SID	Position.SID= Health.SID AND Pos \leq k2	Read	[TAction_start(a), TAction_end(a)]	-	-

Platoon)⁵. The second policy authorizes captains to access the average position of soldiers that are across some border k (here modeled as $Pos \geq k$), as well as their ids, but only during action a. Moreover, this policy states that the average can be computed only on windows of 1 hour with 1 as step. The third policy allows a doctor to monitor the heart beats only of those soldiers belonging to his/her platoon. In contrast, by means of the fourth policy a doctor is able to monitor the average of heart beats of soldiers not belonging to his/her platoon, but only with a window of size and step equal to 1. Finally, the fifth policy states that during action a doctors are authorized to monitor the heart beats of all the soldiers that do not cross some border k2, independent from their platoons.

4 Access Control Policies Semantics

In this section, we introduce the semantics of the access control policies presented in Section 3. In particular, given an access control policy acp in order to define its semantics we need to define the semantics of the subject, protection object, and privilege component, as well as the semantics of window time constraints. We do not provide an explicit semantics for general time constraints since we include it in the protection object specification semantics.

Definition 3. (Subject specification semantics). Let Roles be the set of all possible roles, and Sbj be the set of all possible subjects. We denote with $\lambda(x)$ the set of subjects authorized to play role $x \in \text{Roles}$, and returned by the assignment function $\lambda : \text{Roles} \longrightarrow \text{Sbj}$. Given an access control policy acp , the semantics of the subject specification of acp , is given by the α function, defined as follows:

$$- \alpha(acp) = \lambda(acp \cdot \text{sbj})$$

Definition 4. (Protection object specification semantics). Given an access control policy acp , the protection object specification semantics of acp is the set of tuples denoted by the protection object specification and the general time constraint stated in acp . On the basis of the semantics of Aurora operators, the protection object specification semantics is given by the β function defined as follows:

⁵ We assume that each subject has an associated profile, i.e., a set of attributes modeling the subject's characteristics, like for instance the platoon one belongs to.

- if $|acp.obj.STRs|=1$, then $\beta(acp)=\text{Map}(A_1, \dots, A_n)(\text{Filter}(acp.obj.\text{EXPs} \wedge TS \geq acp.GTC.begin \wedge TS \leq acp.GTC.end)(acp.obj.STRs))$, otherwise
- $\beta(acp)=\text{Map}(A_1, \dots, A_n)(\text{Filter}(acp.obj.\text{EXPs} \wedge TS \geq acp.GTC.begin \wedge TS \leq acp.GTC.end)(\text{Join}(\{S_1, \dots, S_n\}))), S_j \in acp.obj.STRs \forall j \in \{1, \dots n\}$;

where $\{A_1, \dots, A_n\}$ belongs to $acp.obj.ATTs$. If $acp.obj.ATTs = *$, then $\{A_1, \dots, A_n\}$ are all the attributes of streams belonging to $acp.obj.STRs$.

Table 3. Protection object specification semantics for the access control policies in Table 2

AcP	Protection object semantics
acp_1	$\text{Map}(\text{Pos}, \text{SID})(\text{Filter}(\text{Position.Platon}=self.Platon)(\text{Position}))$
acp_2	$\text{Map}(\text{Pos}, \text{SID})(\text{Filter}(TS \geq \text{TAction_start}(a) \wedge TS \leq \text{TAction_end}(a) \wedge \text{Pos} \geq k)(\text{Position}))$
acp_3	$\text{Map}(\text{Heart}, \text{SID})(\text{Filter}(\text{Health.Platon}=self.Platon)(\text{Health}))$
acp_4	$\text{Map}(\text{Heart}, \text{SID})(\text{Filter}(\text{Health.Platon} \neq self.Platon)(\text{Health}))$
acp_4	$\text{Map}(\text{Heart}, \text{SID})(\text{Filter}(TS \geq \text{TAction_start}(a) \wedge TS \leq \text{TAction_end}(a) \wedge \text{Position.SID} = \text{Health.SID} \wedge \text{Pos} \leq k)(\text{Join}(\text{Health}, \text{Position})))$

Example 3. Table 3 presents the protection object specification semantics for the access control policies presented in Table 2. The first row refers to the first access control policy in Table 2 where no general time constraints are specified and the protection object is defined over a single stream, i.e., Position, by projecting the Pos and SID attributes and applying the condition $\text{Position.Platon}=self.Platon$. The semantics of the corresponding protection object is the set of tuples resulting from a Filter operation applied to the Position stream, with expression $\text{Position.Platon}=self.Platon$. Then, to obtain the final set of tuples representing the semantics, a Map operator is applied to the resulting stream projecting the Pos and SID attributes.

By contrast, a general time constraint is specified in the second access control policy in Table 2. In such a case, in addition to the predicate specified in the protection object (i.e., $\text{Pos} \geq k$), the Filter operator also contains the conditions related to the general time constraint (i.e., $TS \geq \text{TAction_start}(a) \wedge TS \leq \text{TAction_end}(a)$).

The semantics of the protection object specification of the third and fourth policies are similar to the first one, since similar to the first access control policy, the third and fourth access control policies do not have any general time constraint, and the protection objects are defined in terms of a single stream. By contrast the protection object specification of the last policy denotes a view generated on two streams (i.e., Health and Position streams). In such a case, the tuples denoting the protection object specification semantics are the result of a join operator applied on Health and Position streams, where no join predicate is specified, thus to obtain the Cartesian product. To the resulting stream a Filter operator is applied with the expressions contained in the protection object specification, that is, the join predicate $\text{Position.SID}=\text{Health.SID}$ and the predicate $\text{Pos} \leq k$. Then, the predicate modeling the general time constraint (i.e., $TS \geq \text{TAction_start}(a) \wedge TS \leq \text{TAction_end}(a)$) is added to the expression. Finally, the Map operation projects the Heart and SID attributes.

The semantics of privilege specification is given by the following definition.

Definition 5. (Privilege semantics). Given an access control policy ACP , the privilege semantics of ACP is given by the χ function defined as follows:

- if $\text{ACP}.\text{priv} = \text{Read}$, then $\chi(\text{ACP}) = \{\text{Filter}, \text{Bsort}, \text{Map}, \text{Union}, \text{Join}, \text{Min}, \text{Max}, \text{Count}, \text{Avg}, \text{Sum}\}$
- $\chi(\text{ACP}) = \text{ACP}.\text{priv}$, otherwise.

Finally, we need to state the semantics of window time constraints.

Definition 6. (Window Time constraint semantics). Given an access control policy ACP , the window time constraint semantics of ACP is given by the δ function defined as follows:

- if $\text{ACP}.\text{priv} \in \{\text{Min}, \text{Max}, \text{Count}, \text{Avg}, \text{Sum}\}$, then $\delta(\text{ACP}) = [\text{ACP}.\text{WTC}.\text{size}, \text{ACP}.\text{WTC}.\text{step}]$;
- $\delta(\text{ACP}) = \text{null}$, otherwise.

We are now ready to define the semantics of an access control policy.

Definition 7. Access control policy semantics. Given an access control policy ACP , the semantics of ACP is defined as follows:

- $\phi(\text{ACP}) = \{(S, T, p, WTC) \mid S = \alpha(\text{ACP}), T = \beta(\text{ACP}), p = \chi(\text{ACP}), WTC = \delta(\text{ACP})\}$

According to Definition 7, given an access control policy ACP we define its semantics as a set of *authorizations* defined as tuples: (S, T, p, WTC) , where S is a subject or a set of subjects identified by the subject specification semantics of ACP , T is a set of tuples identified by the protection object specification semantics of ACP , and p is a set of privileges. If p is an aggregate privilege (i.e., Min, Max, Count, Avg, and Sum) the last component of the policy semantics, i.e., WTC , contains the window time constraint, if any, specified in ACP . An authorization (S, T, p, WTC) states that subjects belonging to S are authorized to exercise privileges p on all and only the tuples belonging to set T , constrained by the window time constraint WTC , if any.

Example 4. Let us consider the second access control policy in Table 2 assuming that in the system the only users to which role *Doctor* is associated are *Alice* and *Bob*. According to Definition 7, the semantics of this access control policy is the set of authorizations (S, T, p, WTC) , where S denotes *Alice* and *Bob*, T is a set of all tuples identified by expression $\text{Map}(\text{Pos}, \text{SID})(\text{Filter}(\text{TS} \geq \text{TAction_start}(a) \text{ AND } \text{TS} \leq \text{TAction_end}(a) \text{ AND } \text{Pos} \geq k)(\text{Position}))$, p is the *Avg* privilege, and WTC denotes the window time constraint, i.e., size and step equal to 1.

5 Conclusion

In this paper, we have presented an access control model and its formal semantics specifically tailored to stream data. The model is role-based and allows the specification of policies at different granularity levels. It supports a set of privileges corresponding to the operations that can be executed over stream data. Moreover, it allows the SA

to specify temporal constraints. This is particularly relevant for aggregate operations where one can constrain the aggregate operation only to specific time intervals.

The work reported in this paper is the first step of a wider project, aiming at developing a complete and fully implemented access control mechanism for stream data. Future work include definition of efficient enforcement strategies for policies specified according to our access control model and the implementation on top of the Borealis data stream engine. Additionally, we plan to extend the presented model to deal with data updates and multiple queries.

References

1. D.J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S.B. Zdonik. The design of the borealis stream processing engine. In *Proceedings of Conference of Innovative Data System Research (CIDR'05)*, pages 277–289, Asilomar, USA, 2005.
2. D.J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S.B. Zdonik. Aurora: a new model and architecture for data stream management. In *VLDB Journal*, 12(2):120–139, 2003.
3. A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. Stream: The Stanford stream data manager. In *Proceedings of ACM SIGMOD'03*, page 665, San Diego, USA, 2003.
4. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '02)*, pages 1–16, New York, USA, 2002.
5. S. Castano, M.G. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison-Wesley, 1995.
6. S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M.A. Shah. TelegraphCQ: continuous dataflow processing for an uncertain world. In *Proceedings of Conference of Innovative Data System Research (CIDR'03)*, Asilomar, USA, 2003.
7. E. Ferrari and B. Thuraisingham. Secure Database Systems. In O. Diaz and M. Piattini editors, *Advanced Databases: Technology and Design*, Artech House, London, 2000.
8. D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. In *ACM Transaction on Information System Security*, 4(3):224–274, 2001.
9. L. Golab and M.T. Ozsu. Issues in data stream management. In *SIGMOD Record*, 32(2): 5–14, 2003.
10. StreamBase Home Page. [http://www.streambase.com//](http://www.streambase.com/).

Protecting Individual Information Against Inference Attacks in Data Publishing

Chen Li¹, Houtan Shirani-Mehr¹, and Xiaochun Yang^{2,*}

¹ Department of Computer Science, University of California at Irvine, CA, USA
{chenli,hshirani}@ics.uci.edu

² School of Information Science and Engineering, Northeastern University, China
yangxc@mail.neu.edu.cn

Abstract. In many data-publishing applications, the data owner needs to protect sensitive information pertaining to individuals. Meanwhile, certain information is required to be published. The sensitive information could be considered as leaked, if an adversary can infer the real value of a sensitive entry with a high confidence. In this paper we study how to protect sensitive data when an adversary can do inference attacks using association rules derived from the data. We formulate the inference attack model, and develop complexity results on computing a safe partial table. We classify the general problem into subcases based on the requirements of publishing information, and propose the corresponding algorithms for finding a safe partial table to publish. We have conducted an empirical study to evaluate these algorithms on real data.

1 Introduction

As many database applications need to publish information on the Web or share information among different organizations, there is an increasing need for these applications to meet their security requirements. Often the data owner needs to protect sensitive information about individuals, such as the disease of a patient, the salary of an employee, or the ethnicity of a customer. On the other hand, given published data, an adversary could use the available information to infer the sensitive information. For example, common knowledge [1], regression models could be used to infer information. From the data owner's perspective, the method he uses to protect the sensitive information depends on what inference technique he believes an adversary could use to do the inference [2].

In this paper we study the problem of protecting sensitive information about individuals when an adversary does inference attacks using data distributions derived from published information. However, there are various ways the adversary can launch the attack. We study how to protect sensitive information about individuals against inference attacks. We focus on inference attacks using *association rules*. Several important factors need to be considered when deciding what additional information needs to be hidden. First, the hidden information

* Supported by the Natural Science Foundation of China under Grant No.60503036, the Fok Ying Tong Education Foundation of China under Grant No. 104027.

depends on the data owner's tolerance on the confidence the adversary can get about the real value of a sensitive entry. Secondly, the owner has to consider the fact that the adversary can do similar inferences by using other properties of this individual as well as a combination of his properties. Being conservative, a data publisher wants to make sure that the adversary does not obtain the real value with a high probability by using any combination of these properties. Thirdly, often there are application-specific requirements when publishing or sharing information. The application could require certain information be released. Furthermore, the application often requires the owner to release as much information as possible.

These challenges motivate our study, in which we make the following contributions: (i) We formulate a data-privacy framework in which a data owner needs to protect sensitive information against inference attacks using association rules; (ii) We study complexity issues of the problem; (iii) We classify the problem into subcases based on the requirements of publishing information, and develop algorithms for these cases; and (v) We conducted an empirical study to evaluate our algorithms on real data sets.

Due to space limitation, we include more results in the full version [13] of this paper.

1.1 Related Work

There have been many studies on data security and privacy. We briefly summarize some related topics in the context of data publishing and sharing.

k-anonymity [3]: The problem is to hide the quasi-identifiers of sensitive entities. Our goal is to hide the real values of individuals, not their identifiers.

Privacy-preserving data mining [4]: The problem is to hide distribution properties of a data set without revealing the information about each individual entity. The most commonly used approach is to perturb the data, say, using randomizing functions. In the applications we are considering, data can only be hidden, but cannot be modified. Evfimievski et al. [5] developed a privacy-preserving framework based on the notion of amplification.

A closely related topic is association-rule hiding (ARH for short) [6, 7], in which the goal is to prevent sensitive association rules from being disclosed. For instance, Atallah et al. [6] proved that it is NP-hard to hide a set of association rules while minimizing the effect on other association rules. This effect is measured as the number of rules lost and the number of rules newly introduced in the hiding process. In [6, 7] several heuristics were proposed to hide association rules. Although our approach is also based on hiding sensitive association rules, there are several differences. (1) Our goal is to hide sensitive entries (the same with [8]), and we use hiding association rules as an intermediate step, while ARH's goal is to hide rules. (2) Most ARH works are dealing with transactional data with values of 1 and 0, while we are dealing with more general data, mostly categorical data. (3) Most ARH works try to minimize the side effect of information hiding on other rules, while we focus on minimizing information loss in terms of the number of hidden entries. (4) We consider application-specific requirements

about what information needs to be published, which are not considered either in ARH works nor [8]. In order to experimentally show the difference between our algorithms and those ARH algorithms, we also implemented one of their approaches to compare the results.

Other related works include the following. [9] developed data-dependent approaches to data privacy. [10] developed an encryption-based approach to access control on publishing XML documents. There are also studies on inference control in relational databases [11], and studies on XML security control (e.g., [12]).

2 Data-Privacy Framework

We consider applications in which a data owner has data stored in a relational table $R(A_1, \dots, A_k)$, which has the following three types of entries.

- A **positive entry** is an entry the data owner has to release, and it is marked with a positive sign (“+”) in the table.
- A **negative entry**, also called a *sensitive entry*, is an entry the owner wants to hide, and it is marked with a negative sign (“-”). The owner hides the value of each negative entry by replacing it with a NULL. A record with a sensitive entry is called a *sensitive record*. In this study we focus on the case where all the negative entries are from the same attribute, called the *sensitive attribute*, denoted by S . Our results can be extended to the case of multiple sensitive attributes.
- An **unmarked entry** is an entry without a sign. It means that the owner can publish its original value, or hide it, in order to protect other sensitive entries against inference attacks.

2.1 Association Rules

A *partial table* is a table in which some entries have been hidden by the data owner. Given a partial table, there are various ways the adversary could do inference. In this work, we focus on the case where the adversary utilizes the derived *association rules* [6,7] from the partial table to do inference. A *pattern* for the relation R is a set of attribute-value pairs: $\{(D_1, v_1), (D_2, v_2), \dots, (D_k, v_k)\}$, in which each D_i is an attribute in the relation (possibly the sensitive attribute), and each v_i is a value for the attribute D_i . The *support* of this pattern p , denoted by $supp(p)$, is the number of records in the table that satisfy the following condition: for each $i = 1, \dots, k$, the value of the record on attribute D_i is v_i . We say such records *have the pattern p*. An *association rule* is in the format of $r : p \rightarrow s$, in which p is a pattern that does not use the sensitive attribute, and s is a value for the sensitive attribute S . We call p the *condition pattern* of the rule r . The *support* of r , denoted by $supp(r)$, is the support of the pattern $p \cup \{(S, s)\}$ (or “ $p \cup \{s\}$ ” for short). The confidence of r is

$$conf(r) = \frac{supp(p \cup \{(S, s)\})}{supp(p)}.$$

Each record with the pattern $p \cup \{(S, s)\}$ is called *relevant* to the rule r .

2.2 Inference Attacks Using Association Rules

Let t be a sensitive record. Its sensitive entry, e , for the sensitive attribute S , is hidden by the data owner. The adversary uses a set of *condition attributes* to infer the original value of the sensitive entry e . Based on the assumption that a set of condition attributes are given, our techniques can provide the corresponding privacy guarantees.

Let $\{D_1, \dots, D_m\}$ be a nonempty subset of the condition attributes of the sensitive record t . Consider the corresponding association rule r : $\{(D_1, t[D_1]), \dots, (D_m, t[D_m])\} \rightarrow t[S]$. Here " $t[D_i]$ " denotes the value of attribute D_i in record t . The adversary could use r to infer the sensitive value $t[S]$ of record t . We assume that the data owner provides a minimum support minsupp and a minimum confidence minconf to specify his tolerance of information leakage. We say that the association rule r *leaks* the sensitive information in record t if (i) the support of r is greater than minsupp , and (ii) the confidence of r is greater than minconf . In this case, we call rule r an *unsafe* association rule. So a rule is safe if its support is within minsupp , or its confidence is within minconf .

For the sensitive record t , there are different subsets of its condition attributes. A sensitive entry is *leaked* if one of these association rules is unsafe. Otherwise, the sensitive entry is *safe*. The data owner, being conservative, wants to make sure that the sensitive entry of this record is not leaked, i.e., none of its association rules is unsafe. A partial table is called *safe* if each of its sensitive entries is safe, i.e., it does not have an unsafe association rule.

In order to make sure that each sensitive entry is safe in a partial table, the data owner needs to decide which of other (unmarked) entries need to be hidden, so that each association rule for the sensitive entries is safe. While there are many such safe partial tables, we are interested in finding one that has the “maximum” amount of information. In this study we consider information amount measured by the number of entries that are not hidden. Intuitively, the more entries that are hidden, the less information is published. A safe partial table T_p is called *optimal* if there is no other partial table whose information amount is greater than that of T_p . In the following sections we study how to compute a safe partial table while releasing as much information as possible. For limited space, we focus on the case where the minimum support is 0. We discuss how to extend the results to the case where $\text{minsupp} > 0$ in [13].

3 Complexity Results

In this section we study complexity issues related to the problem of computing a safe partial table. All the proofs are in [13].

Theorem 1. *The problem of deciding whether there exists a safe partial table is NP-hard.*

Theorem 2. *The problem of computing an optimal safe partial table is NP-hard.*

								<i>Marked positive:</i> +
+ O	O +	+ +	O +	+ O	O O	O O	+ +	<i>Marked negative:</i> -
+ -	+ -	O -	O -	O -	O -	+ -	+ -	<i>Unmarked:</i> O
(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	

Fig. 1. Subcases (the shaded region represents sensitive records)

The complexity results above are developed for the general cases, where the positive entries can appear in any attribute. Often the application requires the positive entries be for certain attributes. Based on where the positive entries are allowed in the table, we study various subcases. A table can be divided into four regions: Condition entries of the nonsensitive records, sensitive entries (those entries of the sensitive attribute) of the nonsensitive records, condition entries of the sensitive records, and sensitive entries of the sensitive records which are always marked negative. Based on whether the application allows each region to be hidden, there are eight subcases as shown in Fig. 1. Besides the negative mark, only entries marked with \bigcirc in Fig. 1 can be hidden.

We study the complexity issues in these subcases. We have proved that the problem of computing an optimal safe partial table is NP-hard for each of the subcases (a)-(g) in Fig. 1. For each subcase, there is always a safe partial table, since there is one unmarked region, and hiding all the entries in this region can make the table become safe. For subcase (h), the problem of computing an optimal solution is not defined since no entry can be hidden.

4 Computing a Safe Partial Table Efficiently

In this section, we study how to compute a safe partial table efficiently. We first analyze the subcases in Fig. 1, and identify two cases, cases (a) and (b), on which our algorithms for other remaining cases are based. We develop an algorithm for finding a solution for case (a). In the next section, we will study case (b). Algorithms for finding solutions of remaining cases are developed in [13].

Lemma 1 tells us that, in cases (f) and (g), we do not need to hide the condition entries of those nonsensitive records to find a solution. As a consequence, an algorithm for case (a) is applicable for case (g), and an algorithm for case (e) is applicable for case (f) as well.

Lemma 1. *In cases (f) and (g), for each safe partial table T that has hidden condition entries of the nonsensitive records, there is another safe partial table T' that does not hide more entries than T , and none of the condition entries of the nonsensitive records is hidden in T' .*

4.1 Algorithm for Case (a)

The main idea of the algorithm is to reduce the problem of finding an optimal safe partial table to the Set Multi-Cover problem [14].

Definition 1. (*Set Multi-Cover*) Given a universal set $\mathcal{U} = \{e_1, e_2, \dots, e_n\}$ and a set \mathcal{L} of some subsets of \mathcal{U} , $\mathcal{L} = \{S_i | S_i \subseteq \mathcal{U}\}$, our goal is to find a smallest sub-collection $\mathcal{C} \subseteq \mathcal{L}$ that covers each element $e \in \mathcal{U}$ at least r_e times.

Consider a sensitive record with a sensitive value s . It has an unsafe rule $r : p \rightarrow s$, whose confidence is greater than the confidence threshold $minconf$. In case (a), to reduce this confidence, we can only hide the sensitive entries of those nonsensitive records. In particular, we need to consider those records relevant to the rule (i.e., they have the pattern $p \cup \{s\}$), and choose some to hide their sensitive entries. The minimum number β_p of such entries that require to be hidden should satisfy the following inequality:

$$\frac{supp(p \cup \{s\}) - \beta_p}{supp(p)} \leq minconf.$$

For each nonsensitive record t with an s value and with at least one of these patterns, let S_t be the set of the patterns that the record t has. S_t is called the *pattern set* of t . Now we convert our problem of finding an optimal partial table to an instance of the set multi-cover problem. Let the universal set \mathcal{U} consist of the condition patterns in the unsafe association rules of the record. Let the set \mathcal{L} consist of the pattern sets of the records with an s value. Our goal is to find a cover for these sets that covers each condition pattern p at least $r_p = \beta_p$ times.

Our algorithm works as follows. In each iteration, we select a set S_t with the largest number of unsafe patterns, and add it to the cover. Correspondingly, we hide the sensitive entry of the record t . Thus we decrease the r_p value of each pattern in the set S_t by one.

The performance ratio of the algorithm is known to be H_k , in which k is the size of largest set, and H_k is the harmonic number of order k [14]. k is at most $2^{(n-1)}$, where n is the number of condition attributes. If \mathcal{C} is the cover obtained by our algorithm, and \mathcal{C}^* is the optimal cover, then

$$\frac{|\mathcal{C}|}{|\mathcal{C}^*|} \leq H_{2^{(n-1)}} < (n-1) \ln 2 + 1.$$

The upper bound for performance ratio is computed using the upper bound for H_k in [15].

Multiple Sensitive Values: For case (a), there could be more than one sensitive value. Notice that since we only hide entries of the sensitive attribute, hiding the entries with one sensitive value does not affect the rules of a sensitive record with a different sensitive value. To see the reason, consider the confidence formula for an association rule $p \rightarrow s$, $conf(p \rightarrow s) = \frac{supp(p \cup \{s\})}{supp(p)}$. Hiding a sensitive value s will only reduce $supp(p \cup \{s\})$, and has no effect on other association rules $p \rightarrow t$, where $t \neq s$. Therefore we can group the sensitive records according to their sensitive values, and run the algorithm above for each group of records to find a safe partial table.

5 Algorithm for Case (b)

In this section we study how to compute a safe partial table for case (b) in Fig. 1, in which we can only hide the condition entries of those nonsensitive records. Similarly to the way we deal with case (a), we first develop an algorithm for the case where the sensitive records have the same sensitive value. We can extend it (13) to the case where the sensitive records can have different sensitive values.

5.1 R-Graph

Consider the case where we have a set $H = \{t_1, \dots, t_m\}$ of sensitive records with the same sensitive value s with k condition attributes. Some of these records are not safe in the original table, i.e., they have unsafe association rules. Our algorithm for finding a safe partial table is based on a directed graph called *R-Graph*. It defines a partial order among the condition patterns of these sensitive records. It is constructed as follows. For each sensitive record t_i , consider its $2^k - 1$ association rules corresponding to the nonempty subsets of the k condition values of t_i . For each rule r , let $p(r)$ be its condition pattern. There is a node, denoted by $v(r)$, in the R-graph, which corresponds to rule r . For example, for a record $\langle a, b, s \rangle$ with the sensitive value s , the graph has three nodes corresponding to the rules $\{a\} \rightarrow s$, $\{b\} \rightarrow s$, and $\{a, b\} \rightarrow s$, respectively.

The graph has a directed edge from a vertex $v(r)$ to $v(r')$ if the pattern $p(r) \supset p(r')$, and $|p(r)| = |p(r')| + 1$. That is, the pattern $p(r)$ is obtained by adding one more value to pattern $p(r')$. We call $v(r)$ a *parent* of $v(r')$. Intuitively, hiding an entry of a record relevant to the pattern of the child node will also reduce the confidence of the rule of the pattern of the parent node.

Each node $v(r)$ in the *R-graph* is associated with two numbers. The first number, called the *MinHide* of the rule r , represents the minimum number of records relevant to the rule r that need to be hidden in order to make rule r safe. By “hidden” we mean each such record should have at least one of its condition values hidden. That is, *MinHide* is the minimum integer x that satisfies:

$$\frac{\text{supp}(p(r) \cup \{s\}) - x}{\text{supp}(p(r)) - x} \leq \text{minconf}.$$

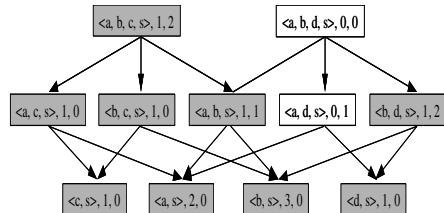
Given a partial table, a node is safe iff its *MinHide* value is 0. The second number, called the *ExactMatch* number of the rule, is the number of records relevant to the rule, but not relevant to the rule of any ancestor of this node.

For instance, let $\text{minconf} = 0.55$. Then the constructed R-graph for the table in Fig. 2(a) is shown in Fig. 2(b). There are two sensitive records $\langle a, b, c, s \rangle$ and $\langle a, b, d, s \rangle$. The graph has 11 nodes corresponding to the 11 different condition patterns of the records, such as $\{a, b, c, s\}$, $\{a, b, s\}$, $\{b, c, s\}$, etc. There is an edge from the node $\{a, b, c, s\}$ to the node $\{a, b, s\}$, since the former can be obtained by adding a value c to the latter. Consider the rule $r : \{b, d\} \rightarrow s$ and the corresponding pattern $p(r) : \{b, d\}$. The node $p(r)$ is associated with two values. The *MinHide* value of the rule is 1, meaning that we need to hide at least one record relevant to this rule, in order to make the rule safe (in the current

ID	A_1	A_2	A_3	A_4
t_1	a+	b+	c+	s ⁻
t_2	a+	b+	c+	s
t_3	a+	b+	c+	s
t_4	a+	b+	*+	s
t_5	a+	b+	d+	s ⁻
t_6	a+	*+	d+	s
t_7	*+	b+	d+	s
t_8	*+	b+	d+	s
t_9	*+	*+	d+	*

Each * represents distinct value that is different from the other values.

(a)



(b)

Fig. 2. An example of a table with two sensitive records and its R-Graph. Each shaded node represents an unsafe rule, i.e., its MinHide value is greater than zero.

partial table). The ExactMatch value of this node is 2, since there are two records relevant to this rule, and not relevant to the rule of any of its ancestors. Notice that as we run the algorithm to hide more entries, these values of the nodes in the graph can change, since we are hiding more entries.

5.2 Algorithm

Our approximation algorithm works iteratively. In each step, it identifies the records of a pattern, then decides their entries to hide. The algorithm modifies the R-graph after each iteration. It repeats until all the nodes in the R-graph are safe.

Choosing Records to Hide. If the ExactMatch number of a node is not zero, we call this node an *active node*. We could choose such a node to hide its relevant records in order to reduce the confidences of its unsafe descendant nodes. The algorithm chooses one of the “top-level” active nodes, i.e., those that do not have active ancestors. For instance, in Fig. 2(b), node $\{a, b, c, s\}$ is a top-level active node. By hiding a condition value of a record relevant to this rule, we can reduce the confidence of the rule, and possibly the confidences of its unsafe descendant nodes. In the case where we have more than one top-level active node, we can use one of the following methods to choose one to hide its entries.

- RANDOM: We randomly select one of them.
- RANDOM-MAX: Among those active top-level nodes with the maximum number of attributes, we randomly select one.
- MAX-DESCENDANT: We choose the node with the largest number of unsafe descendant nodes. Intuitively, hiding a condition value of a record relevant to such a pattern can help reduce the confidence of many unsafe nodes.

Hiding a Condition Entry in Selected Records. When choosing an entry for a record relevant to a node to hide, we want to use this value to reduce the confidence of as many unsafe descendant nodes as possible. Based on this observation, we choose an entry that, among all the entries of this node, appears in the most unsafe descendant nodes of the node.

Deciding the Number of Records to Hide. After selecting a top-level active node $v(r)$ and its entry e , we need to decide the number m of records relevant to this rule whose e value we want to hide. One naive way is to use the ExactMatch value of this rule. This method does not consider the MinHide values of the descendants of this node that share this e value, thus may hide too many records. (Notice that an active node could have a MinHide value of 0.) To solve this problem, we decide value m as follows. We consider all the unsafe descendant nodes of this node $v(r)$ that share this e value, and let α to be the minimum value of these MinHide values. Then we choose m to be the minimum of α and the ExactMatch number of this rule r .

6 Experiments

We have conducted experiments to evaluate our proposed algorithms. Based on the analysis in Section 4, we compare cases (a) and (b) in this paper and cases (c) to (g) in [13]. For case (h), no entries can be hidden, except those sensitive entries. We used two real relational data sets. The first one is the “adult” database from the UC Irvine machine learning repository.¹ The data set has 32,561 records in its training set, and we randomly chose 30,000 of them in our experiments. We used the 7 categorical attributes as the condition attributes, including gender, marital status, level of employment, highest education degree, number of education degrees, and salary (low or high). We used its occupation attribute as the sensitive attribute. The second data set contains information about customers of a bank company. It has 30,000 records. We used 7 categorical attributes: background (award or bad records), loan type (such as house purchase or education), amount of savings (low, medium, or high), occupation, marital status, number of apartments/houses, and resident type. We used the credit rating attribute as the sensitive attribute.

In the experiments, we evaluated how the following factors affect the number of entries hidden by the algorithms and their efficiency. (1) The number of condition attributes; (2) the number of sensitive records; (3) the number of records; (4) the value of the minimum confidence $minconf$; (5) the value of minimum support $minsupp$, and (6) the number of unsafe rules. We also compared our algorithms with one of the association-rule-hiding algorithms called the CR algorithm [7]. All the algorithms were implemented in C++, and run on an Intel Pentium IV 2.4GHz PC with 512MB RAM. For each setting, we tested an algorithm by running it 10 times to compute the average values.

6.1 Algorithm Implementation for Case (a)

We randomly selected a certain number of records from each data set as sensitive records, and let this number vary between 2 and 35. We set the owner’s minimum confidence $minconf = 0.6$, and minimum support $minsupp = 60$, which is 0.2% of the table size.

¹ Downloaded from <http://www.ics.uci.edu/~mlearn/MLSummary.html>

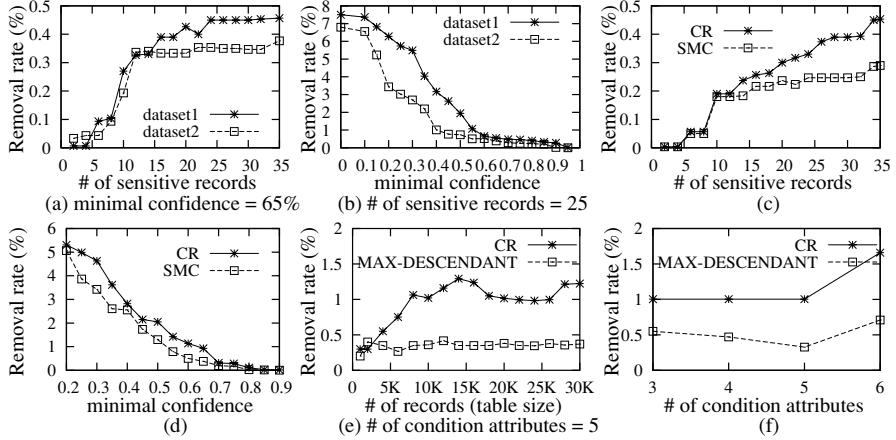


Fig. 3. Test results

It is computationally prohibitive to implement the exhaustive search algorithm. We implemented the algorithm based on Set Multi Cover, called the SMC algorithm, as described in Section 4. Fig. 3(a) shows how the number of sensitive records affects the number of entries hidden by this algorithm. The y -axis, called “Removal rate,” is the percentage of the total number of entries in the table that are hidden. It shows that as the number of sensitive records increased, the number of removals in the sensitive attribute increased, since more records could have unsafe association rules, resulting in hiding more entries. For dataset 1, when there were 4 sensitive entries, the algorithm decided to hide about 0.006% of all the entries. When there were 35 sensitive entries, the algorithm decided to hide about 0.45% of all the entries. Next, we selected 20 different sensitive records, and let the minimum confidence minconf vary from 0 to 1.0. Fig. 3(b) shows that as the minimum confidence increases, there are fewer unsafe rules, resulting in hiding fewer entries.

Comparison with ARH algorithms. We experimentally compared our algorithms with those association-rule-hiding (ARH) algorithms [67]. We chose the CR algorithm [7] as a representative. As explained in Section 1, our problem is different from the problem of hiding association rules in various aspects. The goal of this comparison is to show how many entries we need to remove if we were to use one of these ARH algorithms to solve our problem. When adopting the CR algorithm to our setting, we used it to hide those unsafe rules introduced by the sensitive records. We followed the algorithm’s way of deciding the sequence of hiding the association rules, and the way of choosing records to hide. Since in case (a), we can only remove values of the sensitive attribute, we ignored the last step in the CR algorithm that chooses an entry of a record to hide. Figs. 3(c) and (d) show the results of our SMC algorithm and the CR algorithm for case (a), using the same setting as described above. Fig. 3(c) shows that our SMC

algorithm hides much fewer additional entries than the CR algorithm. Fig. 3(d) shows that the number of removals decreased for both algorithms, as the minimum confidence $minconf$ decreased. Again, our SMC algorithm outperformed the CR algorithm.

6.2 Algorithm Implementation for Case (b)

We also implemented the algorithm described in Section 5 for case (b), in which we can only hide the condition entries of those nonsensitive records. We used data set 1. The minimum confidence $minconf = 0.60$ and the minimal support $minsupp = 60$, which is about 0.2% of the table size. We randomly chose some number of records (table size) and categorical attributes. We increased the number of records and condition attributes. We randomly chose 10 entries (with the same value) as sensitive entries. Three heuristics were implemented to choose a top-level node in the R-Graph, namely, RANDOM, RANDOM-MAX, and MAX-DESCENDANT, as described in Section 5.2. Their results were very similar, so we mainly reported the results of the MAX-DESCENDANT heuristic. In addition, we modified the last step of the CR algorithm slightly so that it can choose a condition entry of a record to hide. Figs. 3(e) and (f) show how the removal rate changed as the number of records in the table increased up to 30,000. As the table size becomes larger, the support of the association rule of the sensitive records also increased, which resulted in more values hidden in order to obtain a safe partial table. Our algorithm (“MAX-DESCENDANT”) removes fewer entries than the CR algorithm.

We next evaluated how the number of condition attributes affects the removal rate. We randomly chose 30,000 records in dataset 1. We set $minsupp = 60$, which is 0.2% of the table size. We let the number of condition attributes vary from 3 to 6. Fig. 3(f) shows that when we increased the number of condition attributes, more values need to be hidden since more condition values can be used to do inference. There are two reasons that our algorithm outperforms the CR algorithm. First, our algorithm chooses a top-level node that can affect more unsafe rules, while CR chooses a record that covers fewer rules, trying to minimize the “side effect” on other rules (e.g., making a safe rule unsafe). Second, our algorithm chooses hiding an entry that occurs in more unsafe descendant patterns of the selected record pattern in the R-graph (i.e., it affects more unsafe rules), whereas CR chooses hiding an entry whose support is maximum in the table. Our experiments [13] show the results when there are multiple sensitive values. We set $minsupp = 60$ and $minconf = 0.7$ for data set 1. We chose 20 sensitive records, and tested the effect of different numbers of sensitive values. We increased the number of different sensitive values from 1 to 9. The results show that when the number of sensitive values increased, we need more removals to get a safe partial table. The reason is that removing conditions of one unsafe rule may increase the confidence of another unsafe rule, which needs more removals to make it safe.

7 Conclusions

In this paper, we studied an important privacy problem in data-publishing or sharing environments: how to protect sensitive information about individuals against inference attacks using association rules? In deciding what data should be released, we need to consider application-specific requirements, e.g., some information has to be released. We formulated the problem, and developed complexity results of the problem. We classified the different cases of the problem, based on what information has to be released. We developed efficient algorithms for computing a safe partial table. We have conducted an empirical study on real data sets to evaluate our techniques.

References

1. Yang, X., Li, C.: Secure XML publishing without information leakage in the presence of data inference. In *VLDB*, 2004.
2. Verykios, V.S., Bertino, E., Fovino, I.N., et al.: State-of-the-art in privacy preserving data mining. *SIGMOD Record*, 33(1):50–57, 2004.
3. Sweeney, L.: k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.
4. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In *SIGMOD Conference*, pages 439–450, 2000.
5. Evfimievski, A., Srikant, R., Agrawal, R., Gehrke, J.: Privacy preserving mining of association rules. In *KDD*, pages 217–228, 2002.
6. Atallah, M., Bertino, E., Elmagarmid, A., Ibrahim, M., Verykios, V.: Disclosure limitation of sensitive rules, 1999.
7. Saygin, Y., Verykios, V.S., Clifton, C.: Using unknowns to prevent discovery of association rules. *SIGMOD Record*, 30(4):45–54, 2001.
8. Aggarwal, C., Pei, J., Zhang, B.: On Privacy Preservation against Adversarial Data Mining. In *Proceedings of ACM SIGKDD 2006*, 2006.
9. Damiani, E., Vimercati, S.D.C.D., Paraboschi, S., Samarati, P.: A Fine-Grained Access Control System for XML Documents. *ACM Transaction on Information and System Security*, 5(2):169–202, 2001.
10. Miklau, G., Suciu, D.: Controlling Access to Published Data using Cryptography. In *VLDB*, 2003.
11. Brodskyand, A., Farkas, C., Jajodia, S.: Secure Databases: Constraints, Inference Channels, and Monitoring Disclosures. *TKDE*, 12(6):900–919, 2000.
12. Bertino, E., Carminati, B., Ferrari, E.: A Secure Publishing Service for Digital Libraries of XML Documents. In *ISC*, pages 347–362, 2001.
13. Li, C., Shirani-Mehr, H., Yang, X.: Protecting Individual Information Against Inference Attacks in Data Publishing. UCI Technical Report, 2006.
14. Rajagopalan, S., Vazirani, V.V.: Primal-dual RNC approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, 28(2):525–540, 1999.
15. Graham, R.L., Knuth, D.E., Patashnik, O.: *Concrete mathematics: a foundation for computer science*. Addison-Wesley Longman Publishing Co., USA, 1989.

Quality Aware Privacy Protection for Location-Based Services

Zhen Xiao^{1,2}, Xiaofeng Meng^{1,2}, and Jianliang Xu³

¹ School of Information, Renmin University of China

² Key Laboratory of Data Engineering and Knowledge Engineering, MOE
{xiaozhen_xfmeng}@ruc.edu.cn

³ Department of Computer Science, Hong Kong Baptist University
xujl@comp.hkbu.edu.hk

Abstract. Protection of users' privacy has been a central issue for location-based services (LBSs). In this paper, we classify two kinds of privacy protection requirements in LBS: *location anonymity* and *identifier anonymity*. While the location cloaking technique under the k -anonymity model can provide a good protection of users' privacy, it reduces the resolution of location information and, hence, may degrade the quality of service (QoS). To strike a balance between the location privacy and QoS, we present a quality-aware anonymity model for protecting location privacy while meeting user specified QoS requirements. In the model, a mobile user can specify the minimum anonymity level requirement upon location privacy as well as the maximum cloaking latency and the maximum cloaking region size requirements upon QoS. In accordance with the model, we develop an efficient directed-graph based cloaking algorithm to achieve both high-quality location anonymity and identifier anonymity. The performance objective is to maximize the cloaking success rate under the privacy and QoS constraints. Furthermore, we introduce an option of using dummy locations to achieve a 100% cloaking success rate at the cost of communication overhead. Experimental results show the effectiveness of our cloaking algorithm under various privacy and QoS requirements.

Keywords: Privacy, Location-based Services, QoS.

1 Introduction

With the advances in wireless communication and mobile positioning technologies, location-based services (LBSs) have become increasingly popular for mobile users. In these applications, mobile users¹ send their location information to service providers and enjoy various types of location-based services, such as mobile yellow page (e.g., “Where is my nearest restaurant”), mobile buddy list (e.g., “Where is my nearest friend”), traffic navigation (e.g., “What is my shortest path to the Summer Palace”), and emergency support services (e.g., “I need help and send me the nearest police”) [12].

¹ In this paper, we use “mobile user”, “mobile client”, and “user” interchangeably.

While people get much benefit from the useful and convenient information provided by LBSs, the privacy threat of revealing a mobile user's personal information (including the identifier and location) has become a severe issue [3][4]. A traditional solution to protecting privacy is the use of pseudonymity [5]. That is, for any LBS request, a trusted middleware is employed to replace the real identifier of the user with a pseudonym before forwarding the request to a service provider [10][11]. However, the location information enclosed in the request may lead to personal re-identification. An attacker can link the location to some particular individual based on external knowledge. For example, if we know the location exclusively belongs to some owner, the corresponding request can thus be linked to the location owner [8][9].

In general, there are two kinds of privacy protection requirements in LBS:

- **Location anonymity.** This is to protect a user's location from being disclosed when the location information is sensitive (e.g., in a clinic or pub). A common technique is to cloak the user's location by an extended region. Under the k -anonymity model [6], the region is large enough such that it contains at least $k - 1$ other users.
- **Identifier anonymity.** This is to hide a user's identifier when the message content is sensitive (e.g., political or financial data). Again location cloaking can be applied to provide identifier anonymity. Under the k -anonymity model [6], user locations are cloaked such that a location is covered by at least $k - 1$ other requests. In this way, a request is not identifiable from the other $k - 1$ requests.

While the k -anonymity model can provide a good protection of users' privacy, it reduces the resolution of the location information and, hence, may degrade the quality of service (QoS). It is often desirable to strike a balance between the location privacy and QoS requirements. In this paper, we present a quality-aware anonymity model for protecting location privacy while meeting user specified QoS requirements. In our model, a mobile user can specify the following requirements in each LBS request: 1) the minimum anonymity level k , indicating the location cloaking should satisfy both k -location-anonymity and k -identifier-anonymity; 2) the maximum cloaking latency Δt , representing the maximum cloaking delay that the user can tolerate; 3) the maximum cloaking region size δ , indicating the maximum tolerable error in location data. While k reflects the user's requirement upon location privacy, Δt and δ represent the user's QoS requirements. Since the privacy/QoS tradeoff for a user may change over time under different circumstances, we allow these requirements to vary from one request to another even for the same user.

In accordance with the quality-aware anonymity model, we develop an efficient directed-graph based cloaking algorithm to perform anonymization over LBS requests. The performance objective is to maximize the cloaking success rate with privacy protected and QoS guaranteed. Furthermore, we introduce an option of using dummy locations to achieve a 100% cloaking success rate at the cost of communication overhead. Under this scenario, we would like to make use of as few dummies as possible to minimize the communication overhead. We conduct

a series of experiments to evaluate the effectiveness of the proposed algorithms. The results show that our algorithms are superior under various privacy and QoS requirements.

The rest of this paper is organized as follows. We first review some related work on protecting location privacy in Section 2. In Section 3, we describe our quality-aware anonymity model. An efficient cloaking algorithm is proposed in Section 4. Some discussions and improvements are presented in Section 5. Section 6 presents the implementation and experimental results of our proposed algorithms. Finally, Section 7 concludes the paper.

2 Related Work

Several different models have been used for protecting location privacy. Kido *et al.* [2] proposed a dummy-based approach, in which a user sends the actual location with several fake locations (“dummies”) to a service provider. The service provider processes and returns an answer for each received location. The user finally refines the result based on the actual location.

The k -anonymity model was originally introduced for privacy protection in conventional database applications [7]. As defined in [6], a release of data provides k -anonymity protection if the information for each individual contained in the release cannot be distinguished from at least $k - 1$ individuals whose information also appear in the release. In the context of LBS, Gruteser and Grunwald [10] first adopted the k -anonymity model and proposed a quad-tree based cloaking algorithm. They assume a static anonymity requirement k_{min} for all users. To achieve k -anonymity, the algorithm recursively subdivides the area around a user’s location into four quadrants until the number of users in the area falls below k_{min} , and then returns the previous quadrant as the cloaking region. This technique does not differentiate the privacy requirements of different users. Moreover, no restriction is imposed on the cloaking region size. Thus, a cloaking region can be very large, which may lead to an inaccurate query result and a poor service quality.

Gedik and Liu [11] recently proposed the technique of supporting personalized privacy requirements, capturing the privacy and QoS requirements on a per-user basis. A location cloaking algorithm called *CliqueCloak* was developed. CliqueCloak constructs an undirected graph for all the requests that have not been anonymized yet. Each time the server receives a new request, it attempts to identify a clique involving the new request and some existing requests, and cloak them together with the same region. However, this method has several drawbacks. First, the effectiveness of this method is limited to users with small k values (i.e., 2-5). As shown in [11], we can hardly find the anonymity set for requests with larger k values. Second, the cost of searching a clique in a graph is costly. Third, some requests that cannot be anonymized will be dropped when their lifetimes expire. This will affect the user experience towards the service. Different from [11], our proposed cloaking algorithm considers the tradeoff between privacy and QoS requirements to achieve a higher cloaking success rate.

A different framework named *Casper* was proposed in [13]. Casper employed a grid-based pyramid structure to index all user locations. Besides the anonymity level k , a user can specify A_{min} , indicating that the user wants to hide the location information within an area of at least A_{min} . This model has the following concerns. First, k and A_{min} have a similar functionality. In fact, the higher is the value of k , the larger is the cloaking area. Second, the cloaking region may expand arbitrarily large if k is set to a large value and few users present nearby. To address this problem, Casper uses a privacy-aware query processor to return a list of candidate query results to the anonymizing proxy, who has to locally refine the actual result from the candidate list. This approach incurs a high query processing cost, a high communication cost, and a high local computation cost. In contrast, to reduce such costs, we enforce some temporal and spatial QoS requirements when performing location cloaking.

3 System Model

We consider a LBS system consisting of mobile clients, a trusted anonymizing proxy, and LBS providers [10,11]. Upon a user query, the mobile client first sends the LBS request to the anonymizing proxy through an authenticated and encrypted connection. The request consists of the user's identifier id , current location $l = (l.x, l.y)$, current time t , as well as the service related content such as the query (denoted by $data$). Additionally, the mobile client can specify in the request its privacy and QoS requirements, which include the desired anonymity level k , the tolerable maximum cloaking delay Δt , and the acceptable maximum cloaking region size (denoted by a radius of δ). Thus, a request from user i is defined as: $r_i = (id, l, k, \Delta t, \delta, data, t)$.

Based on the request's privacy and QoS requirements, the anonymizing proxy expands the location l into a cloaking region \mathcal{L} (to be detailed later in this section). Moreover, the identifier id is replaced with a pseudonym id' (e.g., a secure hash number). The original request is transformed into a new anonymized request, $r'_i = (id', \mathcal{L}, data)$, and is forwarded to the LBS provider. Finally, the anonymized request is processed by LBS provider. The query result is sent back to the anonymizing proxy, which, after refining the result, returns the final result to the mobile client.

We adopt the k -anonymity model [6] for protecting location privacy. Given a set of user requests $\{r_1, r_2, \dots, r_n\}$ and their anonymized requests $\{r'_1, r'_2, \dots, r'_n\}$, the location k -anonymity model is defined as follows:

Definition 1. *For any request r_i , the location k -anonymity is satisfied if and only if 1) r_i 's cloaking region $r'_i \cdot \mathcal{L}$ covers the locations of at least $k - 1$ other requests (i.e., $|\{j \mid r_j.l \in r'_i \cdot \mathcal{L}, 1 \leq j \leq n, j \neq i\}| \geq k - 1$) and, 2) r_i 's location $r_i.l$ is covered by the cloaking regions of at least $k - 1$ other requests (i.e., $|\{j \mid r_i.l \in r'_j \cdot \mathcal{L}, 1 \leq j \leq n, j \neq i\}| \geq k - 1$).*

By this definition, a LBS provider cannot distinguish a user's location $r_i.l$ from $k - 1$ other users' locations since they all present in the cloaking region $r'_i \cdot \mathcal{L}$.

Moreover, even knowing the location of a user, a LBS provider cannot tell which request is made by this user since there are k requests all covering this user's location. As such, both location anonymity and identifier anonymity are achieved. We refer to the set of users achieving location anonymity as *location anonymity set* and the set of users achieving identifier anonymity as *identifier anonymity set*. For example, Figure 1 shows four LBS requests from different users as well as their cloaking regions. Since r_1 's cloaking region covers r_1, r_2 , and r_3 , the location anonymity set of r_1 is $\{r_1, r_2, r_3\}$. On the other hand, r_1 is covered by the cloaking regions of r_1 through r_4 . Thus, the identifier anonymity set of r_1 is $\{r_1, r_2, r_3, r_4\}$.

In summary, for any request r and its anonymized request r' , we specify the privacy and QoS requirements from the following three aspects:

1. **Location Privacy.** This requires to expand the user location l into a cloaking region \mathcal{L} such that the k -anonymity model (Definition 1) is satisfied.
2. **Temporal QoS.** This states that the request must be anonymized before the predefined maximum cloaking delay (i.e., $t + \Delta t$).
3. **Spatial QoS.** This specifies that the cloaking region size should not exceed a threshold, i.e., the cloaking region must be inside a circle Ω centered at l and with a radius of δ (i.e., $\mathcal{L} \subseteq \Omega(l, \delta)$).

In general, a larger Δt (or δ) provides more flexibility in location anonymization but results in an extended query delay (or a less accurate query result).

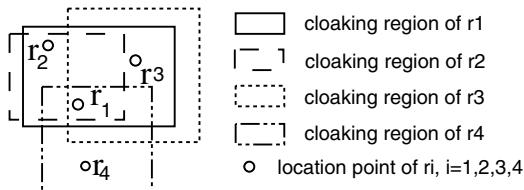


Fig. 1. Illustration of Location Anonymity Set and Identifier Anonymity Set

4 Basic Anonymization Algorithm

The anonymization algorithm turns the user location into a cloaking region based on the privacy and QoS constraints. In this section, we discuss the following problems faced by the anonymization algorithm: 1) when to anonymize which request? and 2) given a request under the cloaking region size constraint, how to find other requests (i.e., the location anonymity set and identifier anonymity set) to satisfy the location k -anonymity model?

Bearing in mind that our objective is to maximize the cloaking success rate, we shall delay the anonymization process of a request until right before its deadline (i.e., $t + \Delta t - \epsilon$), where ϵ is a small time offset set to be the worst cloaking time. In this way, not only can more potential requests join a request's anonymity set, but also other requests have a higher chance to include this request in their anonymity sets.

To tackle the second problem, the CliqueCloak algorithm proposed in [11] constructs an undirected graph to represent the correlations of all requests. The graph is defined as: $G_u = (V, E_u)$, where V is the set of the nodes, each representing a request r received at the anonymizing proxy, and E_u is the set of edges, each representing the neighborship between the corresponding nodes (requests). For any pair of nodes $r_i, r_j \in V$, there exists an edge $e_{ij} = (r_i, r_j) \in E_u$ if and only if the distance between the two requests $|r_i r_j|$ is not more than both $r_i.\delta$ and $r_j.\delta$ ($|r_i r_j| \leq r_i.\delta$ and $|r_i r_j| \leq r_j.\delta$), which indicates the locations of r_i and r_j are within each other's predefined maximum cloaking region size. For any request r_i , its location anonymity set and identifier anonymity set are the same $r_i.U$ that includes all its neighbors except those have k larger than $r_i.k$ and those have less than $k - 1$ neighbors in $r_i.U$. A clique of at least k requests is identified if they share the same anonymity set. These requests are then anonymized with the same cloaking region (the minimum bounding rectangle (MBR) of their location points) at the same time. This approach incurs a huge searching cost for identifying cliques with the worst time complexity of $O(N^2N)$, where N is the number of nodes in the subgraph of $r_i.U$. Moreover, with a restricted definition of the anonymity set, the cloaking success rate is low.

Recall that under our location k -anonymity model, each request can have a different anonymity set and hence a different cloaking region. Thus, in contrast to [11], we build a directed graph rather than an undirected graph. In the directed graph $G_d = (V, E_d)$, for any pair of nodes $r_i, r_j \in V$, there exists an edge $e_{ij} = (r_i, r_j) \in E_d$ from r_i to r_j , if and only if the distance between the two requests is not more than $r_i.\delta$ ($|r_i r_j| \leq r_i.\delta$), which indicates r_j 's location is within r_i 's predefined maximum cloaking region size. Similarly, there exists an edge $e_{ji} = (r_j, r_i) \in E_d$ from r_j to r_i , if and only if r_i 's location is within r_j 's predefined maximum cloaking region size. The location anonymity set of a request r_i , is formed by all its outgoing neighbors $r_i.U_{out}$, i.e., $r_i.U_{out} = \{r_i\} \cup \{r_j \mid (r_i, r_j) \in G_d(V, E_d)\}$, and the identifier anonymity set, is formed by all its incoming neighbors $r_i.U_{in}$, i.e., $r_i.U_{in} = \{r_i\} \cup \{r_j \mid (r_j, r_i) \in G_d(V, E_d)\}$. For each request r_i , we maintain a *flag* to identify its status, i.e., *flag* = *unanonymized* means r_i has not been anonymized, and *flag* = *forwarded* means r_i has been anonymized successfully and forwarded to the service provider but not yet deleted in the graph. A request r_i can be anonymized immediately if there are at least $k - 1$ other anonymized requests in $r_i.U_{out}$ (i.e., $|\{j \mid r_j \in r_i.U_{out}, r_j.flag = forwarded, j \neq i\}| \geq k - 1$) and $k - 1$ other anonymized requests in $r_i.U_{in}$ (i.e., $|\{j \mid r_j \in r_i.U_{in}, r_j.flag = forwarded, j \neq i\}| \geq k - 1$).² The cloaking region of r_i is then represented by the MBR of the location points of the requests in the location anonymity set (denoted by $MBR(r_i.U_{out})$).

We use an example to illustrate the differences between CliqueCloak (Figure 2(a)) and our proposed cloaking algorithm (Figure 2(b)). The same set of user requests with corresponding k values are shown in the figure, where

² Initially, no requests are flagged as “forwarded”. We employ CliqueCloak to anonymize requests in the warm-up period; our proposed cloaking algorithm follows after the warm-up period.

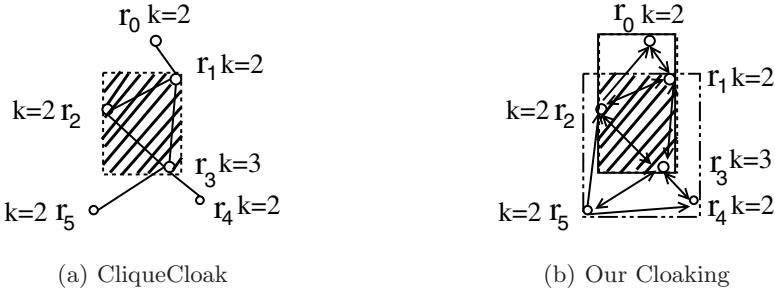


Fig. 2. An Example: the differences between CliqueCloak and our proposed cloaking

they are numbered in the ascending order of their deadlines. We assume r_0 has been anonymized successfully ($r_0.flag = forwarded$ in our cloaking). With CliqueCloak, r_1 , r_2 , and r_3 form a clique and are cloaked with the same region (their MBR, represented by the shaded area in Figure 2(a)). Then they will be deleted from the graph. Next, r_4 and r_5 will be dropped because they cannot find enough neighbors. With our proposed cloaking algorithm, the neighborships between the requests are different and some new edges are added in the directed graph. First, r_1 is processed with $U_{out} = U_{in} = \{r_0, r_1, r_2, r_3\}$. Because $r_0.flag = forwarded$, satisfying $r_1.k = 2$, we can anonymize r_1 with cloaking region $MBR(r_0, r_1, r_2, r_3)$. Then, r_2 is processed with $U_{out} = \{r_0, r_1, r_2, r_3\}$ and $U_{in} = \{r_1, r_2, r_3, r_5\}$. Because r_0 and r_1 have been anonymized successfully, satisfying $r_2.k = 2$, we can also cloak it with $MBR(r_0, r_1, r_2, r_3)$. Similarly, all the other requests will be successfully anonymized with $r'_3.\mathcal{L} = MBR(r_1, r_2, r_3, r_4, r_5)$, $r'_4.\mathcal{L} = MBR(r_3, r_4)$, $r'_5.\mathcal{L} = MBR(r_2, r_3, r_4, r_5)$. Obviously, by allowing different cloaking regions for different requests, our proposed cloaking algorithm gets a higher success rate than CliqueCloak. Moreover, as shown in Figure 2(b), the cloaking regions of r_1 , r_2 , and r_3 cover some more requests in addition to r_1 , r_2 , and r_3 , thereby providing a higher privacy level than CliqueCloak. In the following, we describe the detailed data structures and related algorithms for anonymizing user requests.

4.1 Data Structures

As mentioned, we employ a dynamic in-memory directed graph for all user requests. To facilitate the construction and maintenance of the graph, we build a spatial index (i.e., R-tree) over the location points ($r_i.l$'s) of all requests. Thus, we can use a window query to quickly find the neighbors of a request. Additionally, we maintain a min-heap to order the requests according to their cloaking deadlines (i.e., the key is $r_i.t + r_i.\Delta t$).

4.2 Algorithms

Maintenance: Algorithm 1 details the maintenance of the data structures. Given a new incoming request r_i , we first update the spatial index and the

heap. Next we update the graph. We start by searching the spatial index using a range query with $r_i.l$ as the central point and δ_{max} as the radius, where δ_{max} is the maximum cloaking region size requirement of all the requests. The requests in the search result C are the candidates for being r_i 's neighbors in the graph. Each r_j in C is filtered based on whether the distance between r_j and r_i is within $r_i.\delta$ or $r_j.\delta$. In the former case, we construct an edge from r_i to r_j . In the latter case, we construct an edge from r_j to r_i . In both cases, they are added to each other's outgoing neighbor set U_{out} and incoming neighbor set U_{in} . In the algorithm, $r_i.n$ is used to represent the cardinality of $U_{in} \cup U_{out}$.

Algorithm 1: Maintenance ($r_i = \{id, x, y, t, \Delta t, k, \delta, data\}$)

```

1 insert  $r_i$  into the spatial index and the heap;
2 create a new node for  $r_i$  in the graph;
3  $r_i.n \leftarrow 0$ ;  $r_i.U_{in} \leftarrow \{r_i\}$ ;  $r_i.U_{out} \leftarrow \{r_i\}$ ;
4  $C \leftarrow$  a range query  $Q$  on the spatial index,
 $Q = ((x - \delta_{max}, y - \delta_{max}), (x + \delta_{max}, y + \delta_{max}))$ ;
5 forall  $r_j \in C, r_j \neq r_i$  do
6   if  $|r_i.r_j| \leq r_i.\delta$  or  $|r_i.r_j| \leq r_j.\delta$  then
7     if  $|r_i.r_j| \leq r_i.\delta$  then
8       create an edge  $(r_i, r_j)$  in the graph;
9        $r_i.U_{out} \leftarrow r_i.U_{out} \cup \{r_j\}$ ;  $r_j.U_{in} \leftarrow r_j.U_{in} \cup \{r_i\}$ ;
10    if  $|r_i.r_j| \leq r_j.\delta$  then
11      create an edge  $(r_j, r_i)$  in the graph;
12       $r_i.U_{in} \leftarrow r_i.U_{in} \cup \{r_j\}$ ;  $r_j.U_{out} \leftarrow r_j.U_{out} \cup \{r_i\}$ ;
13     $r_i.n \leftarrow r_i.n + 1$ ;  $r_j.n \leftarrow r_j.n + 1$ ;
14  end
15 end

```

Cloaking Algorithm: Algorithm 2 describes the cloaking algorithm. The input request r is the first request approaching its deadline. We first compare r 's cloaking time constraint $r.t + r.\Delta t$ with current time t_{now} . If $r.t + r.\Delta t - t_{now} \leq \varepsilon$, where ε is a small time offset set to be the worst cloaking time, we cloak r immediately. Otherwise, we delay the anonymization of r to the time $r.t + r.\Delta t - \varepsilon$. In the cloaking algorithm, we compute the number of r 's neighbors that have been anonymized successfully in the outgoing neighbor set and the incoming neighbor set, denoted by k_o and k_i respectively. If both k_o and k_i satisfy r 's minimum anonymity level ($k_o \geq r.k - 1$ and $k_i \geq r.k - 1$), r can be cloaked as $r' = (pid, MBR(r.U_{out}), r.data)$ and its flag is set as “forwarded”, where pid is the pseudonym that replaces $r.id$. Otherwise, the cloaking fails and the request is deleted from the graph. This algorithm has a time complexity $O(n)$, where n is the number of r 's neighbors.

After r is successfully cloaked, we delay the removal of r in the graph until all its neighbors have been processed. This is because otherwise the neighborships between r and its neighbors will disappear. And when we anonymize r 's neighbors later, they cannot include r in their anonymity sets and, hence, reduce

the privacy level and the cloaking success rate. Thus, we then decrease the unprocessed neighbors (n) of each r 's neighbor r_j in $r.U_{out} \cup r.U_{in}$. If r_j has been anonymized already before r and r is the last neighbor of r_j to be processed, we can remove r_j from the graph. If all neighbors of r have been anonymized before it, we can remove r from the graph. No matter whether the cloaking succeeds or fails, finally the request r should be removed from the spatial index and the heap.

Algorithm 2: Cloaking

```

1 get the top request  $r$  in the heap;
2 if  $t + \Delta t - t_{now} < \varepsilon$  then
3   forall  $r' \in r.U_{out}$  do if  $r'.flag = forwarded$  then  $k_o++$ ;
4   forall  $r' \in r.U_{in}$  do if  $r'.flag = forwarded$  then  $k_i++$ ;
5   if  $k_o \geq k - 1$  and  $k_i \geq k - 1$  then
6     send out  $R = (pid, MBR(r.U_{out}), r.data)$  for  $r$ ;
7      $r.flag = forwarded$ ;
8     forall  $r' \in r.U_{out} \cup r.U_{in}, r' \neq r$  do
9        $r'.n \leftarrow r'.n - 1$ ;
10      if  $r'.n = 0$  and  $r'.flag = forwarded$  then delete  $r'$  from the graph;
11    end
12    if  $r.n = 0$  then delete  $r$  from the graph;
13  else
14    delete  $r$  from the graph;
15  delete  $r$  from the spatial index and the heap;
16 end

```

5 Improvement with Dummy Requests

This section discusses an improvement by using dummy requests in case of a cloaking failure. With dummies introduced, we can guarantee a successful cloaking for every request and a 100% success rate. Therefore, we only need to maintain for each node r the number of incoming neighbors $r.k_i$ and the number of outgoing neighbors $r.k_o$ ($r.flag$ is no longer maintained). If both $r.k_i$ and $r.k_o$ satisfy the required privacy level $r.k$, the cloaking can proceed with success. Otherwise, we use Algorithm 3 to generate enough dummies such that the dummies and the real neighbors together form r 's anonymity set. The time complexity of this cloaking algorithm is $O(1)$.

We generate dummies for a request r based on the following requirements. First, dummies should be within both the location anonymity set and the identifier anonymity set such that the privacy level will be higher. Second, dummies must be indistinguishable from actual requests. Third, dummies should satisfy the spatial QoS requirement of r . Thus, to avoid expanding the existing cloaking region, the location of each dummy distributes randomly within $MBR(r.U_{out})$. The cloaking region of each dummy request, d , which also cover the location point of r , is a random spatial region between $MBR(\{r, d\})$ and $MBR(r.U_{out})$. As such, the dummies and r become both incoming neighbors and outgoing neighbors, and the service provider will have difficulty in identifying dummies.

Algorithm 3: Dummy (x, y, k_d, M)

```

1 compute the MBR of  $M$ ,  $L = ((x_{min}, y_{min}), (x_{max}, y_{max}))$ ;
2 for  $i = 1$  to  $k_d$  do
3    $x \leftarrow random(x_{min}, x); x' \leftarrow random(x, x_{max});$ 
4    $y \leftarrow random(y_{min}, y); y' \leftarrow random(y, y_{max});$ 
5    $\mathcal{L}_i \leftarrow ((x, y), (x', y'));$ 
6   send out the  $i$ th cloaked dummy request,  $R_i \leftarrow (pid, \mathcal{L}_i, data);$ 
7 end

```

Algorithm 3 shows the detailed dummy generation process. The inputs of the procedure are: location $(l.x, l.y)$ of request r to be cloaked, the number of dummies to be generated, and r 's outgoing neighbor set as M . The pseudonym and service related content are also randomly generated. Finally, we send the cloaked dummy requests out to the service provider.

6 Experiments

In this section, we experimentally compare the effectiveness of our cloaking algorithm against CliqueCloak [11] under various location privacy and QoS settings. In all the experiments, we use Thomas Brinkhoff Network-based Generator of Moving Objects [2] to generate a set of moving objects. The input to the generator is the road map of Oldenburg County (with an area of about $200km^2$). We simulate 20,000 moving objects that are uniformly distributed in the spatial space at the initial time and afterwards continuously move on the road network. These moving objects issue LBS requests with their current locations at a query interval of 20,000 s. In each request, Δt , k , and δ are assigned uniformly between the range $[.05 - .15]\%$ of the update interval (i.e., 1,000-3,000), [2 – 5] users, and [.01-0.05]\% of the space (i.e., 2-10), respectively. We set the worst cloaking time offset ε as 10 s.

Recall that the goal of our cloaking algorithm is to maximize the number of requests anonymized successfully in accordance with their privacy and QoS requirements. We first evaluate the cloaking success rate with various privacy and QoS requirements. Figures 3(a), 3(b), and 3(c) show the effect of varying k , δ , and Δt on the success rate, respectively. In all cases tested, our method without using dummies always outperforms CliqueCloak, by 5-25% in terms of cloaking success rate. By using dummy requests, we can even achieve a 100% success rate. In Figures 3(a), both CliqueCloak and our method show that the requests with larger k values are more difficult to anonymize, thus getting a lower success rate. However, the performance degradation of our method is less significant than that of CliqueCloak. This indicates that our method is more robust for larger k values. Figures 3(b) and 3(c) show that a larger δ or Δt improves the flexibility in location anonymization and thus getting a higher success rate.

We then measure the efficiency of our location-anonymity model in terms of users' privacy requirements. The relative location anonymity level is measured

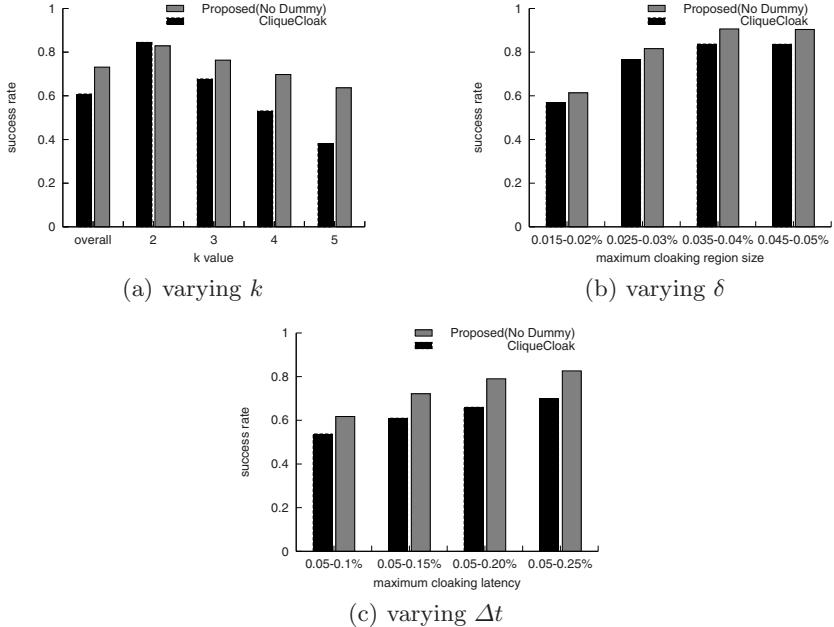


Fig. 3. Performance of Cloaking Success Rate under Different Settings

by k'/k , where k' is the number of users actually included in the cloaking region while k is the user required number. In Figure 4, we compare the relative anonymity level of our method against CliqueCloak under different k values. In our method, by using dummies, the relative anonymity level can be up to 9 for $k = 2$, meaning that the requests are actually anonymized with $k \approx 18$ on average. Without using dummies, the relative anonymity level is from 5.2 for $k = 2$ to 2.5 for $k = 5$, meaning that the requests are actually anonymized with $k \approx 10$ on average. CliqueCloak provides a lower level from 1.2 for $k = 2$ to 1.0 for $k = 5$. This result also demonstrates that even without dummies our method can support larger k values up to 10 while CliqueCloak is limited to smaller k values. In Figure 5, we measure the portion of dummy requests generated in the total requests under varying k values. Requests with larger k require more

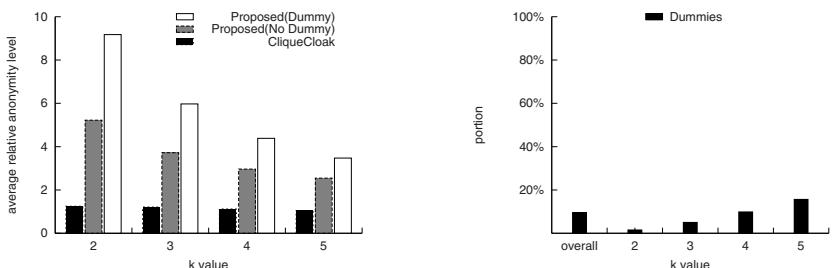


Fig. 4. Relative Anonymity Level

Fig. 5. Portion of Dummy Requests

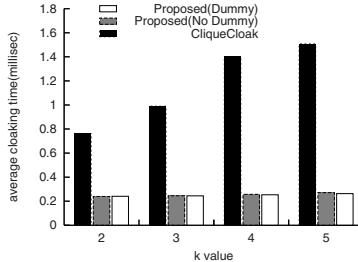


Fig. 6. Cloaking Efficiency

neighbors and hence a higher percent of dummies. On average, we can achieve a 100% success rate with about 10% dummies (and thus 10% of communication overhead), which we think is acceptable.

Finally, Figure 6 shows the effect of k on the cloaking efficiency. In all cases tested, our method shows a much shorter cloaking time than CliqueCloak. When k increases, (more neighbors are formed), the average cloaking time lengths as a result of increasing cost of searching anonymity sets. However, the performance degradation of our method is much less smaller than that of CliqueCloak.

7 Conclusion

In this paper, we have discussed the problem of quality-aware privacy protection in location-based services. We classified the privacy requirements into location anonymity and identifier anonymity. To protect both of these two anonymities, we have presented a quality-aware k -anonymity model that allows a mobile user to specify in each LBS request the location privacy requirement as well as the temporal and spatial QoS requirements. We have developed an efficient directed-graph based cloaking algorithm to achieve a high cloaking success rate while satisfying the privacy and QoS requirements. Moreover, we have introduced the use of dummy requests to achieve a 100% cloaking success rate at the cost of communication overhead. Experimental evaluation have verified the effectiveness of our model and the proposed cloaking algorithms under various privacy and QoS requirements.

Acknowledgments

This research was partially supported by the grants from the Natural Science Foundation of China under grant number 60573091, 60273018; Program for New Century Excellent Talents in University(NCET). Jianliang Xu's work was supported by grants from the Research Grants Council, Hong Kong SAR, China (Project Nos. HKBU 2115/05E and HKBU 2112/06E).

References

1. R. José, N. Davies. Scalable and Flexible Location-Based Services for Ubiquitous Information Access. In Proceedings of First International Symposium on Handheld and Ubiquitous Computing, 1999.
2. J. Schiller and A. Voisard, editors. Location-Based Services. Morgan Kaufmann Publishers, 2004.
3. L. Barkhuus and A. K. Dey. Location-Based Services for Mobile Telephony: a Study of Users' Privacy Concerns. In INTERACT, 2003.
4. A. R. Beresford and F. Stajano. Location Privacy in Pervasive Computing. IEEE Pervasive Computing, 2(1):46-55, 2003.
5. A. Pfitzmann and M. Hansen. Anonymity, Unlinkability, Unobservability, Pseudonymity, and Identity management - A Consolidated Proposal for Terminology, 2005.
6. L. Sweeney. K-anonymity: A model for protecting privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10(5):557C570, 2002.
7. P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical Report SRI-CSL- 98-04, SRI International, 1998.
8. M. Gruteser and B. Hoh. On the Anonymity of Periodic Location Samples. In SPC, 2005.
9. A. Machanavajjhala, J. Gehrke, and D. Kifer. l-Diversity: Privacy Beyond k-Anonymity. In ICDE, 2006.
10. M. Gruteser and D. Grunwald. Anonymous usage of location based services through spatial and temporal cloaking. In ACM/USENIX MobiSys, 2003.
11. B. Gedik and L. Liu. Location Privacy in Mobile Systems: A Personalized Anonymization Model. In ICDCS, 2005.
12. H. Kido, Y. Yanagisawa, and T. Satoh. Protection of Location Privacy using Dummies for Location-based Services. In ICPS, 2005.
13. M. F. Mokbel, C. Chow and W. G. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In VLDB, 2006.
14. T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. GeoInformatica, 6(2):153C180, 2002.

Implementation of Bitmap Based Incognito and Performance Evaluation*

Hyun-Ho Kang, Jae-Myung Kim, Gap-Joo Na, and Sang-Won Lee

Sungkyunkwan University, Suwon, Korea
`{hanpaldduk, jam02, factory, swlee}@skku.edu`

Abstract. In the era of the Internet, more and more privacy-sensitive data is published online. Even though this kind of data are published with sensitive attributes such as name and social security number removed, the privacy can be revealed by joining those data with some other external data. This technique is called joining attack. Among many techniques developed against the joining attack, the k-anonymization generalizes and/or suppresses some portions of the released microdata so that no individual can be uniquely distinguished from a group of size k . Incognito is one of the most efficient k-anonymization algorithms. However, Incognito requires many repeating sorts against large volume data. In this paper, we propose a bitmap based Incognito algorithm. Using the bitmap technique, we can completely eliminate the expensive sort operations, and can even prune some steps in the traditional Incognito algorithm. Therefore, our new algorithm can improve the performance by an order of magnitude. From the perspective of implementation, the key issue in bitmap based Incognito is the speed of bitwise AND/OR and bit-count operations. For this, we designed and implemented a bitmap package which exploits the Single Instruction Multiple Data technique. Our experimental result shows that bitmap-based Incognito outperforms the traditional Incognito by an order of magnitude.

1 Introduction

In the era of the Internet, more and more privacy-sensitive data is published online. In general, this kind of data is provided without attributes such as name and social security number, for privacy. In some cases, however, the privacy can be revealed by joining those data with some other external data, and this technique is called joining attack [2]. Among many techniques against the joining attack, the k-anonymization generalizes and/or suppresses some portions of the released microdata so that no individual can be uniquely distinguished from a group of size k [3]. For example, see below table 1 and 2. If we join table 1 with table 2 using the columns of Birthdate, Sex and Zipcode, we can easily know that Andre has a disease ‘Flu’. On the other hand, if we join table 1 with the table 3, we know that ‘Andre’ has either ‘Flu’ or ‘Broken Arm’, but we could not

* This research was supported in part by MIC, Korea under ITRC IITA-2006-(C1090-0603-0046), in part by MIC & IITA through IT Leading R&D Support Project.

know the exact disease of ‘Andre’. In summary, k-anonymity guarantees that k data items are returned so that join attackers are not able to know the exact individual value of privacy sensitive data item.

Table 1. Voter Registration Data

Name	Birthdate	Sex	Zip
Andre	1/21/76	M	53715
Beth	1/10/81	F	55410
Carol	10/1/44	F	90210
Dan	2/21/84	M	02174
Carol	4/19/72	F	02237

Table 2. Hospital Patient Data

Birthdate	Sex	Zip	Diseases
1/21/76	M	53715	Flu
4/16/86	F	53715	Hepatitis
2/28/76	M	53703	Brochitis
1/21/76	M	53703	Broken Arm
4/13/86	F	53706	Sprained Ankle
2/28/76	F	53706	Hang Nail

Table 3. Generalized Hospital Patient Data

Birthdate	Sex	Zip	Diseases
1/21/76	M	537**	Flu
4/16/86	F	537**	Hepatitis
2/28/76	M	537**	Brochitis
1/21/76	M	537**	Broken Arm
4/13/86	F	537**	Sprained Ankle
2/28/76	F	537**	Hang Nail

In general, the cost of a k-anonymity algorithm determines by lattice construction cost and k-anonymity check cost for each node in lattice. Binary search algorithm [7] was proposed for the lattice construction for k-anonymity. This is a traditional algorithm for k-anonymization before introducing Incognito. The algorithm uses the observation that if no generalization of height h satisfies k-anonymity, then no generalization of height $h' < h$ will satisfy k-anonymity. If the maximum height in the generalization lattice is h , it begins by checking each generalization at height $\lfloor h/2 \rfloor$. If a generalization exists at this height that satisfies k-anonymity, the search proceeds to look at the generalizations of height $\lfloor h/4 \rfloor$. Otherwise, it searches the generalizations of height $\lfloor 3h/4 \rfloor$, and so forth. This algorithm is proven to find a single minimal full-domain k-anonymization according to this definition. However, it has at least two limitations. First, binary search needs fully sized lattice for k-anonymity. However, in order to construct the lattice for calculating k-anonymity, it should scan the whole base data repeatedly. A large number of the expensive join operations are required to build the lattice, and to be worse, for each subset, the number of joins increases according to the number of elements in the subset. Another problem is that it supports only a fixed subset. It could not support k-anonymity for join attacks involving dynamic subsets. In summary, binary search approach is not a viable option for k-anonymity, especially when the data size is large.

Incognito algorithm resolves these problems. While constructing a lattice, Incognito only considers the nodes which have survived from the previous ($n-1$ step) subsets, and thus, compared to binary search approach, Incognito can dramatically reduce the cost of lattice construction. In this respect, the contribution of Incognito is comparable to the Apriori data mining algorithm [6] which drastically reduces the number of candidate sets to be considered when mining frequent item sets. Lattice construction cost influence performance because Incognito and binary search have same k -anonymity check cost per node. However, Incognito itself is still inefficient in checking whether each node in the lattice satisfies the k -anonymity because it requires expensive sort operations against large volume data.

In this paper, we propose a bitmap based Incognito algorithm. Bitmap based Incognito can completely eliminate expensive sort operations, and can even prune some steps in the traditional Incognito algorithm. Therefore, our new algorithm can improve the performance by an order of magnitude. From the perspective of implementation, the key issue in bitmap based Incognito is the speed of bitwise AND/OR and bit-count operations. For this, we designed and implemented a bitmap package which exploits the Single Instruction Multiple Data (SIMD) technique [4].

The contributions of this work can be stated as follow. First, even though we use the same framework of Incognito in building the lattice, we further improve its performance by adopting a bitmap-based technique for checking whether a node satisfies the k -anonymity and thus eliminating the expensive sort operations. Second, we can achieve further performance optimization by pruning some steps in the Incognito, and this optimization is possible because our algorithm is based on bitmap. Finally, we design and implement a bitmap package which fully exploits the SIMD technique in order to accelerate the core bitmap operations.

2 Basic Definitions and Incognito

In this section, we provide basic terminologies necessary to understand the remainder of this paper, and also introduce the idea, basic algorithmic framework of Incognito and its problems.

2.1 Basic Definition

The following definitions are not developed by the authors, but they are cited from the original Incognito paper [1]:

- **Quasi-Identifier(QI) Attribute Set:** A quasi-identifier attribute set Q is a minimal set of attributes in table T that can be joined with external data to re-identify each individual record [2].
- **Frequency Set:** Consider relation T and a QI attribute set Q with n attributes. The frequency set of T with respect to Q is a mapping from each unique combination of values $\langle q_0, q_1, \dots, q_n \rangle$ of Q in T (a value group) to the total number of tuples in T with these values of Q (its count).

- **K-Anonymity Property:** Relation T is said to satisfy the k-anonymity property (or k-anonymous) with respect to attribute set Q if every count in the frequency set of T with respect to Q is greater than or equal to k.
- **Generalization [5]:** Generalization is a high dimension value of current dimensions. For example, 5370* is a generalization of 53703 and 53706 in table 2. $A <_D B$ denotes A is a generalization of B. Another example is 537** is generalization of both 5370* and 5371*. There are two types of generalization. One type is direct generalization. The relationship between 5370* and 53703 is a direct generalization. The other type is indirect generalization. The relationship between 537** and 53703 is an indirect generalization. Figure 1 shows a generalization lattice for Zipcode [1].

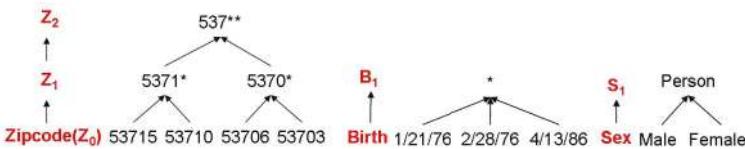


Fig. 1. Single(1-subset) Generalization

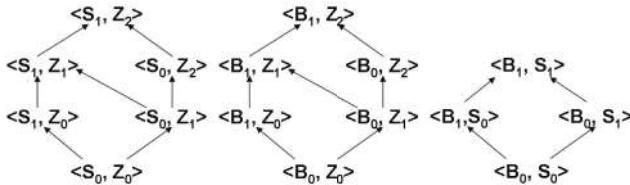
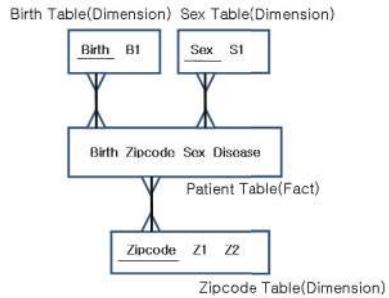


Fig. 2. Generalization lattice for the 2-subset

Before closing this section, we would like to mention two related issues. The first issue is how to represent a relational database for k-anonymity problem [1]. There are two types of relational representation. First, all generalization values are in a row. This representation has a space problem. In figure 3, first and second rows have the same generalization data (*, *, 5371*, 537**). This representation reduces the number of joins because the table has all data. But it has a space overhead which is a duplicate data. Second representation is a star schema in figure 4. We apply 3-Normalization into the first one. As a result of, it needs fewer spaces than the first representation. Second representation consists of one fact table and several dimension tables. In this paper, we assume the normalized relational representation.

The second issue is how to compute the frequency set in SQL. Using the standard SQL, the frequency set can be obtained from T with respect to a set of attributes Q by issuing a COUNT(*) query, with Q as the attribute list in

Birth	Sex	Zipcode	Disease	B1	S1	Z1	Z2
1/21/76	Male	53715	Flu	*	*	5371*	537**
4/13/86	Female	53715	Hepatitis	*	*	5371*	537**
...
2/28/76	Female	53706	Hang	*	*	5370*	537**

Fig. 3. Relational table representation**Fig. 4.** Star schema representation

the GROUP BY clause. For example, in order to check whether the Patients table in Table 2 is 2-anonymous with respect to $\langle \text{Sex}, \text{Zipcode} \rangle$, we issue a query `SELECT COUNT(*) FROM Patients GROUP BY Sex, Zipcode`. Since the result includes groups with count fewer than 2, Patients is not 2-anonymous with respect to $\langle \text{Sex}, \text{Zipcode} \rangle$.

2.2 Incognito

Assume that you want to get k-anonymity of 1-subset (e.g. $\langle \text{Birth} \rangle$, $\langle \text{Sex} \rangle$, $\langle \text{Zipcode} \rangle$), 2-subset ($\langle \text{Birth}, \text{Sex} \rangle$, $\langle \text{Birth}, \text{Zipcode} \rangle$, $\langle \text{Sex}, \text{Zipcode} \rangle$) and 3-subset ($\langle \text{Birth}, \text{Zipcode}, \text{Sex} \rangle$). When using binary search, you need to build all full sized lattice. If the number of quasi-identifier attributes is large, the lattice building cost will be overhead. If each column of QI has l, m, n generalization, there are 12 nodes in a 3-subset lattice (figure 12-b). However, Incognito can drastically reduce the cost of lattice building because it, instead of generating all the candidate nodes of n-subset from the scratch, generate the candidate nodes of n-subset from the nodes in (n-1)-subset which satisfy the k-anonymity. Therefore, Incognito need not consider a large number of nodes which is safely considered not to be k-anonymous (figure 7-a). For this, Incognito exploits the following properties, and this intuition is the main contribution of the Incognito paper [1].

- **Generalization Property:** Let T be a relation, and let P and Q be sets of attributes in T such that $DP <_D DQ$. If T is k-anonymous with respect to P , then T is also k-anonymous with respect to Q .
- **Rollup Property:** If attribute set P is a generalization of Q , counts grouped by P can be computed directly from the counts grouped by Q .
- **Subset Property:** Let T be a relation, and let Q be a set of attributes in T . if T is k-anonymous with respect to Q , then T is k-anonymous with respect to any set of attributes P such that $\subseteq Q$.

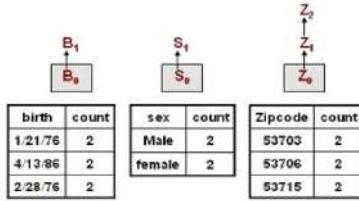


Fig. 5. Check k-anonymity on the 1-subset

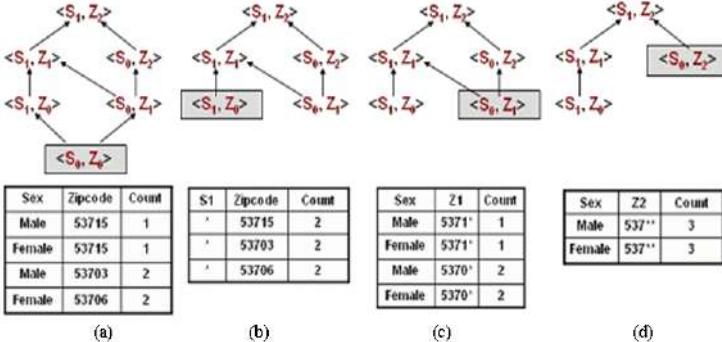


Fig. 6. Check k-anonymity on the 2-subset

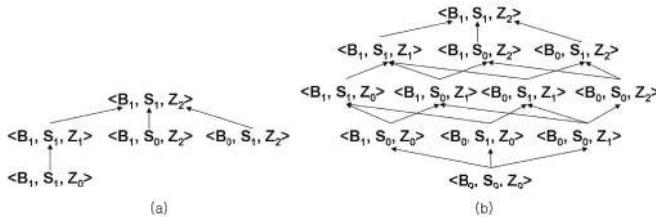


Fig. 7. The 3-subset lattice. (a) The 3-attribute graph generated from 2-attribute results. (b) The 3-attribute lattice that would have been explored without a priori pruning. (e.g. binary search).

1. Assume that $k=2$. First, obtain the frequency sets on each QI and then check whether each frequency set is greater than value k (figure 5). If every frequency set is greater than k , then nodes which were can be used in 2-subset lattices (figure 6).
2. Node $\langle S0, Z0 \rangle$ is removed from a lattice because its frequency set is smaller than k . Node $\langle S1, Z0 \rangle$ is checked which is a direct generalization of $\langle S0, Z0 \rangle$ (figure 6-a, 6-b).
3. The check for both $\langle S1, Z1 \rangle$ and $\langle S1, Z2 \rangle$ can be skipped because the node $\langle S1, Z0 \rangle$ satisfies the k -anonymity. This is due to generalization property.

4. Check whether the nodes $\langle S0, Z1 \rangle$ and $\langle S0, Z2 \rangle$ satisfy the k-anonymity, and we can $\langle S0, Z1 \rangle$ from the lattice because it is not k-anonymous. Thus, we can obtain the intermediate lattice as in figure 6-d.
5. Repeat the same test against the $\langle B, Z \rangle$ and $\langle B, S \rangle$ lattice respectively.
6. Finally, by combining the all remainders in 2-subset lattices, we can obtain a 3-subset lattice as in figure 7-a.

We can publish at passed generalization level after testing all lattices. These results guarantee k-anonymity.

When compared to binary search algorithm, Incognito has at least two advantages. First, as we noted before, while binary search algorithm needs fully sized lattice to test on each subset, Incognito has a less build cost because it only uses nodes which were passed from previous subset. Second, it supports all k-anonymity from 1-subset to n-subset which can be attacked.

Nevertheless, Incognito itself has still a performance problem because it is mainly based on sorting when checking k-anonymity of nodes in lattice. In order to check whether a node satisfy the k-anonymity, it uses a SQL query in the form of `SELECT COUNT(*) FROM(temp) table GROUP BY column`. In general, many relational database systems implement group-by and count operation using the internal sorting [8]. If Incognito can reduce or avoid the sorting operations, then it will be much faster than now. In particular, the size of data used in k-anonymization is too large to fit in main memory, and thus the sort operation will invoke external sort algorithm.

Someone has a question that all data can be loaded into memory, and then sort it. Sometimes this question can be true. However, generally all data could not be loaded into main memory at once. In addition to, we need a different sorting level. Therefore, almost all of test needs a sort. However some nodes can avoid sorts because of rollup property.

3 Bitmap-Based Incognito

In this section, we propose generalization and node generation using bitmap and novel algorithm called bitmap-based; and then, we explain advantage of bitmap-based Icognito algorithm.

3.1 Generalization and Node Generation Using Bitmap

Generalization is upper category of current value. Therefore, it contains at least one sub-category. Assume that, there are some bitmaps and then do bitwise OR operation on them. This result includes them. This is same as generalization. In other words, bitwise OR operation is same as generalization. See table 2. Zipcode of row 3, 4 and 5, 6 are 53703, 53706 respectively. Therefore bitmaps for row 3, 4 and 5, 6 are 001100 and 000011. Generalization of 53703 and 53706 is a 5370*. Its bitmap is a 001111. Bitmap of 5370* can be obtained by using bitwise OR into bitmap for 53703 and 53706. ($001100 \parallel 000011 = 001111$).

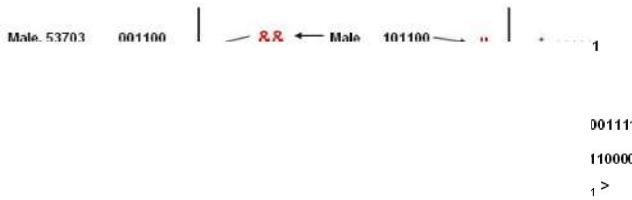


Fig. 8. Generalization and making root nodes of lattices

In mathematic, intersection means that a value is included both A and B. Similarly, generation of node also can be obtained by using bitwise AND operation. See table 2. Person who is a male and 53715 (Zipcode) is in a row 1. Bitmap for male is 101100 and 53715 is 110000. Do bitwise AND into bitmap for male and 53715. Its result is same the bitmap for male and 53715 ($\langle S_0, Z_0 \rangle$). That is, we can build n-subset node by bitwise-ANDing two (n-1)-subset.

3.2 Bitmap-Based Incognito Algorithm

We can get a new Incognito algorithm by applying theory described in section 3.1. Bitmap-based Incognito algorihtm (figure 9) and example are like below.

- 1) Generate bitmaps on quasi-identifier attribute sets;
- 2) Check k-anonymity on the 1-subset;
- 3) LOOP UNTIL current subset \leq subset user want DO
 - 4) Create bitmap of root nodes;
 - 5) LOOP UNTIL there is a test node in the lattice DO
 - 6) Perform a test;
 - IF frequency set is larger than k THEN
 - Assign skip marks to direct/indirect generalization nodes;
 - Decide a test node;
 - ELSE
 - Decide a test node which is a direct generalization;

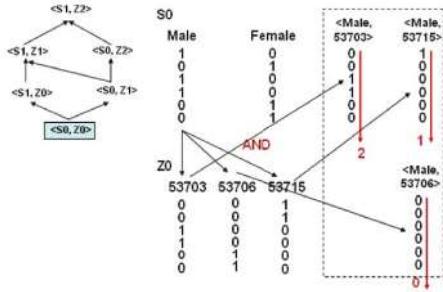
Fig. 9. Bitmap-based Incognito algorithm

1. Check k-anonymity on 1-subset by using bit-count. (figure 10) Every frequency set is greater than k therefore they can be used making 2-subset lattices. Make 2-subset lattices by using bitwise AND. (figure 11)
2. Check frequency set of $\langle S_0, Z_0 \rangle$ by using bit-count. This node is removed from a lattice because its frequency set is smaller than k. Next test node $\langle S_1, Z_0 \rangle$ is generated by using bitwise OR.
3. $\langle S_1, Z_1 \rangle$ and $\langle S_1, Z_2 \rangle$ are can be skipped because $\langle S_1, Z_0 \rangle$ satisfies the k.

4. Does a test on $\langle S0, Z1 \rangle$ and $\langle S0, Z2 \rangle$. As a result of, $\langle S0, Z1 \rangle$ is removed from a lattice. $\langle S0, Z1 \rangle$ and $\langle S0, Z2 \rangle$ were generated by using bitwise OR into bitmap of $\langle S0, Z0 \rangle$ and $\langle S0, Z1 \rangle$ respectively. We can obtain the lattice like a figure 6-d.
5. Does a test on lattice $\langle B, Z \rangle$ and $\langle B, S \rangle$.
6. Make a 3-subset lattice by bitwise AND then test on it. (figure 7-a)

Rowid	Birth			Sex		Zipcode		
	1/21/76	2/28/76	4/13/86	Male	Female	53703	53706	53715
1	1	0	0	1	0	0	0	1
2	0	0	1	0	1	0	0	1
3	0	1	0	1	0	1	0	0
4	1	0	0	1	0	1	0	0
5	0	0	1	0	1	0	1	0
6	0	1	0	0	1	0	1	0

count: 2 2 2 3 3 2 2 2

Fig. 10. Check a 1-subset by bit-count**Fig. 11.** Generation of a root node in a lattice by using bitwise AND

3.3 Advantage of Bitmap Incognito

With the traditional Incognito, the k-anonymity test of each node requires a sort operation over a large data set, although you may use a temp table for exploiting the Rollup property. However, if we employ the bitmap representation for the base data set, there are no or less physical reads because it is a small size and can be compressed compactly. Another advantage is followings. We can get generalization, generation of root nodes and confirmation of frequency can be obtained by bitwise OR/AND and bit-count respectively. Therefore, it does not need access to tables.

4 Optimization Techniques

In this section, we propose three optimization techniques in bitmap-based Incognito algorithm. These are 1-level, reusing, and pruning optimization.

4.1 1-Level Optimization

One disadvantage of bitmap based Incognito is a space overhead. If you have many QI, then you need more space for bitmaps. However, this overhead can be solved by reusing 1-level (1-subset) bitmaps. While generating nodes, we can get them by using bitwise ANDs into 1-level bitmaps. In figure 12, we only bitwise AND into a_2, g_2 and, e_1 to get $\langle a_2, g_2, e_1 \rangle$. However this optimization has a problem which is an increment of number of bitwise AND.

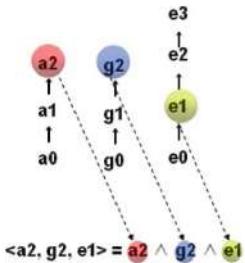


Fig. 12. 1-level optimization

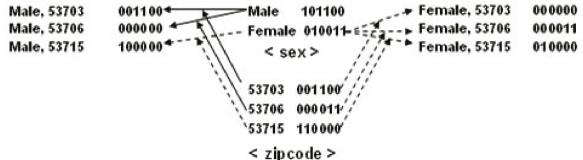


Fig. 13. Pruning optimization

4.2 Reusing Optimization

The 1-level optimization is good at space requirement, but it increases the number of the bitwise AND operations, resulting in performance degradation. We can solve this problem by reusing child bitmaps which are temporarily stored in the previous step. For example, assume that you want to get a bitmap of $\langle a2, g2, e1 \rangle$. If you use the 1-level optimization, you must perform two bitwise AND operations for $\langle a2 \wedge g2 \wedge e1 \rangle$. However, if you temporarily store child bitmaps of $\langle a2, g2 \rangle$ and $\langle g2, e1 \rangle$, you can obtain the bitmap of $\langle a2, g2, e1 \rangle$ by just doing $\langle a2, e1 \rangle \wedge \langle g2, e1 \rangle$ (one bitwise AND). Also, this optimization can be used for generalization. In following case, this optimization is good for performance than 1-Level optimization. There are many columns in QI attribute sets or value k is very small. In this situation, it needs more tests. That is, it uses more bitwise operations.

4.3 Pruning Optimization

If we use the traditional Incognito, it is impossible to decide whether the frequency set is greater than k when the sort operation completes. With the bitmap based Incognito, however, we know the exact value of a frequency set after bitwise AND/OR operation. After bitwise AND/OR, if the element value of frequency set is less than value k, we know that this node is not to be k-anonymous, and thus we can skip the next steps. For example, we want to get $\langle S0, Z0 \rangle$. Do bitwise AND between male and 53703 and also do it between male and 53706 respectively. We can know that $\langle \text{male}, 53703 \rangle$ is greater than k but $\langle \text{male}, 53706 \rangle$ is not. As a result of, we know that this node $\langle S0, Z0 \rangle$ does not support k-anonymity. That is, following bitwise ANDs($\langle \text{male}, 53715 \rangle$, $\langle \text{female}, 53703 \rangle$, ... etc) are can be skipped (dotted arrows) (figure 13).

5 Performance Evaluation

Our experiment data sets are small and big `census.dat` [9]. The size of small and big data sets are about 5MB and 60MB. Small and big experiment data set have QI attribute set which are consist of four columns. These columns are generalized

into 3, 3, 2 and 4 levels respectively. In addition, we build composite index on these columns in the fact table because sorts can be avoided by using index data which is in leaf blocks. These index size are about 2 and 16MB. Ratios are about 40% and 27% compare to their base tables. Our implementation environments are Pentium 4 2.0GHz, 1GB memory, 120GB(7200 rpm), Oracle 10g Release 1 and Intel C++ Compiler 9.0.

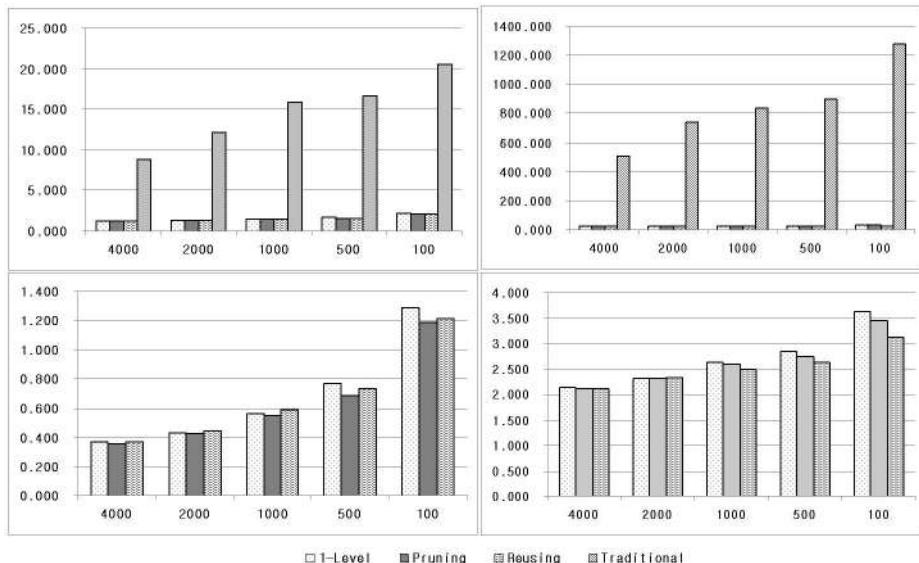


Fig. 14. Performance Evaluation (x and y axis mean value k and time(sec) respectively)

See above results. bitmap based Incognito much faster than traditional one although it includes bitmap creation time. There are many reasons. First bitmaps are much smaller than fact(real) table or index. The size of bitmaps are about 200KB and 2MB respectively. It is much smaller than fact(real data) table or index. Therefore, all operations(generalization and generation of nodes) can be done in main memory. Also, all operations do not need any access into tables. In addition, bitwise operations are very fast than other operations.

Almost all of sort need full table scan although there is B*tree index which is Oracle served. Because each node need different sorts(GROUP BY). If index includes many columns, it can be bigger than base table.

See above results. If k is a low value, time is increased because there are lots of nodes to be tested which are greater than k. For this reason,lattices have more nodes than lattices with high k. It means lattices with low k need more time to finish test. We proposed three types of optimizations. Normally, reusing optimization outperforms among of them because it reduce number of bitwise operations. If user use 1-Level optimization in 4-subset, there are three numbers

bitwise operations at each test phase. When user use the reusing optimization, there is just one number bitwise operation. However, n-1 optimization needs more spaces to maintain bitmaps than others.

6 Conclusion

When compared to traditional k-anonymity algorithms, including binary search, Incognito is very innovative in that it reduces the number of nodes to be considered in building a lattice for the k-anonymity check. However, it is still inefficient in checking the k-anonymity for each node because it is based on expensive sort operations over a large volume of data. In this paper, we proposed the bitmap based Incognito, which is based on bitwise AND/OR and count operations, rather than expensive sorts. In addition, our bitmap-based Incognito comes with some optimizations techniques for pruning some nodes for the k-anonymity check. We show that our approach can improve the performance of the traditional Incognito by an order of magnitude.

References

1. K. LeFevre, D. J. DeWitt and R. Ramakrishnan: “Incognito: efficient full-domain k-anonymity” In Proceedings of the ACM SIGMOD international conference on Management of data, Baltimore, Maryland (2005) 49–60
2. L. Sweeney: “K-anonymity: A model for protecting privacy”, International Journal on Uncertainty, Fuzziness, and Knowledge-based Systems; **10(5)** (2002) 557–570
3. P. Samarati and L. Sweeney: “Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression”, Technical Report SRI-CSL-98-04, SRI Computer Science Laboratory (1998)
4. J. Zhou and K. A. Ross: “Implementing database operations using SIMD instructions”, In Proceedings of the ACM SIGMOD international conference on Management of data, Madison, Wisconsin (2002) 145–156
5. P. Samarati: “Protecting respondents’ identities in microdata release”, IEEE Transactions on Knowledge and Data Engineering **13(6)** (2001) 1010–1027
6. R. Agrawal and R. Srikant: “Fast Algorithms for Mining Association Rules in Large Databases”, In Proceedings of Proceedings of the 32nd International Conference on Very Large Data Bases, Santiago de Chile, Chile (1994) 487–499
7. Roberto J. Bayardo , Rakesh Agrawal, “Data Privacy through Optimal k-Anonymization”, Proceedings of the 21st International Conference on Data Engineering (2005) 217–228
8. Jonathan Lewis, *Cost-Based Oracle Fundamentals*, Apress (2005)
9. Test Data from <http://vldb.skku.ac.kr/mbar/files/>

Prioritized Active Integrity Constraints for Database Maintenance

Luciano Caroprese, Sergio Greco, and Cristian Molinaro

DEIS, Univ. della Calabria, 87030 Rende, Italy
`{lcaroprese, greco, cmolinaro}@deis.unical.it`

Abstract. The paper presents a logic framework wherein constraints and preferences are used for database maintenance and querying. Our proposal is based on the use of a special type of integrity constraints (called *Prioritized Active Integrity Constraints (PAICs)*), whose body defines a constraint on data, whereas the head contains a set of partially ordered actions, which should be performed, if the body constraint is not satisfied, to make consistent the database. Therefore, a preference relation among repairs is introduced on the base of the (partially ordered) actions specified in the head of PAICs. On the base of the preference relation a set of preferred repairs is identified and preferred answers are derived from the database instances which have been made consistent by means of preferred repairs. The paper shows that databases with universal prioritized constraints admit (preferred) repairs and consistent answers if the set of constraints is satisfiable. The paper also shows how PAICs can be rewritten into disjunctive Datalog programs so that repairs can be obtained from the computation of stable models.

1 Introduction

Integrity constraints are generally used to define semantic constraints on data (functional dependencies, inclusion dependencies, exclusion dependencies, etc.) and their enforcement ensures a semantically correct state of a database [1]. In many situations the presence of inconsistent data cannot be avoided. A typical situation arise when two or more consistent data sources are integrated into a single database [18].

The presence of inconsistent data can be managed by repairing the database, i.e. by providing consistent databases, obtained by applying a minimal set of update operations, or by consistently answering queries posed over an inconsistent database.

The following example shows a situation where inconsistencies occur.

Example 1. Consider the database schema consisting of the relation schemas $emp(Name, Dept)$ and $dept(Name)$ with a referential integrity constraint stating that a department appearing in the relation emp must occur in the relation $dept$ too. This constraint can be defined through the first order formula:

$$\forall(E, D) [emp(E, D) \supset dept(D)]$$

Consider now the inconsistent instance $\mathcal{DB} = \{ \text{emp}(john, cs), \text{emp}(john, deis), \text{dept}(deis) \}$. A consistent (repaired) database can be obtained by applying a minimal set of update operations (insertions or deletions); specifically, there exists two possible repairs: R_1 obtained by inserting the tuple $\text{dept}(cs)$ and R_2 obtained by deleting the tuple $\text{emp}(john, cs)$. The answer to the query looking for the name of departments consists of two sets containing the certain answer $deis$ and the uncertain answer cs . \square

The motivation of this work stems from the observation that as an inconsistent database can be repaired in different ways, it is natural to express preferences among the possible actions which make the database consistent.

Example 2. Consider the database of Example 1. It can be repaired either by inserting the missing department tuple $\text{dept}(cs)$ or by deleting the employee tuple $\text{emp}(john, cs)$. In this scenario, suppose that the insertion of a missing department is preferable to the deletion of an existing employee. This preference can be expressed by means of the following prioritized active integrity constraint:

$$\forall(E, D) [\text{emp}(E, D), \text{not dept}(D) \supset +\text{dept}(D) \succ -\text{emp}(E, D)]$$

As R_1 repairs the database by inserting the tuple $\text{dept}(cs)$ whereas R_2 deletes the tuple $\text{emp}(john, cs)$, R_1 is preferable to R_2 . Therefore we have a unique preferred repair and the certain answer to the query of Example 1 contains both department $deis$ and cs . \square

The novelty of the presented approach consists in the formalization of *Prioritized Active Integrity Constraints (PAICs)*, a flexible and easy mechanism for specifying the “preferred” updates, i.e. the actions that should be performed if an integrity constraint is not satisfied. More specifically, prioritized active integrity constraints are constraints which allow us to define the actions to be performed if a constraint is violated and also to introduce a partial order on the actions. The paper shows that databases with universally quantified prioritized constraints admit preferred repairs and (preferred) consistent answers if the set of constraints is satisfiable. The paper also presents how the computation of repairs and consistent answers can be done by rewriting constraints into a disjunctive Datalog program and computing the stable models of the target program; every repair can be obtained from a stable model of the Datalog program and vice versa.

Related work. In the last few years, there have been several proposals considering the computation of repairs and queries over inconsistent databases [2, 3, 16, 17, 22]. Logic-programming-based approaches have been proposed in [16, 17, 4].

The increased interest in preferences in logic programs is reflected by an extensive number of proposals and systems for preference handling. Most of the approaches propose an extension of Gelfond and Lifschitz’s extended logic programming by adding preference information [10, 12, 20, 23]. The literature distinguish *static* and *dynamic* preferences. Static preferences are fixed at the time a theory is specified, i.e. they are “external” to the logic program [20], whereas dynamic preferences appear within the logic program and are determined “on the fly” [5, 10].

The most common form of preference consists in specifying a strict partial order on rules [10,12,20,23], whereas more sophisticated forms of preferences also allow us to specify priorities between conjunctive (disjunctive) knowledge with preconditions [5,20]. In [9] the framework of consistent query answer [2] is extended by allowing preferences among tuples to be expressed. Several families of preferred repairs (i.e. subsets of repairs selected with priorities) have been also investigated.

Organization. The paper is organized as follows: Section 2 gives basic definitions on logic languages; Section 3 presents integrity constraints and recalls the formal definition of repair and consistent answer; Section 4 presents prioritized active integrity constraints; finally, in Section 5 conclusions are drawn.

2 Background

Familiarity with disjunctive logic programs and disjunctive deductive databases is assumed [11,14]. A term is either a constant or a variable. An atom is of the form $p(t_1, \dots, t_h)$, where p is a *predicate symbol* of arity h and t_1, \dots, t_h are terms. A literal is either an atom A or its negation $\text{not } A$. A (*disjunctive Datalog*) rule r is a clause of the form¹:

$$\bigvee_{i=1}^p A_i \leftarrow \bigwedge_{j=1}^m B_j, \bigwedge_{j=m+1}^n \text{not } B_j, \varphi \quad p + n > 0$$

where $A_1, \dots, A_p, B_1, \dots, B_n$ are atoms, while φ is a conjunction of built-in atoms of the form $u \theta v$ where u and v are terms and θ is a comparison predicate. The disjunction $\bigvee_{i=1}^p A_i$ is the *head* of r (denoted by $\text{Head}(r)$), while the conjunction $\bigwedge_{j=1}^m B_j, \bigwedge_{j=m+1}^n \text{not } B_j, \varphi$ is the *body* of r (denoted by $\text{Body}(r)$). It is assumed that each rule is safe, i.e. a variable appearing in the head or in a negative literal also appears in a positive body literal. The expression $H \leftarrow B_1 \vee \dots \vee B_n$ can be used as shorthand for the rules $H \leftarrow B_1, \dots, H \leftarrow B_n$.

A (*disjunctive Datalog*) program is a finite set of rules. A *not-free* (resp. \vee -free) program is called *positive* (resp. *normal*). The *Herbrand Universe* $U_{\mathcal{P}}$ of a program \mathcal{P} is the set of all constants appearing in \mathcal{P} , and its *Herbrand Base* $\mathcal{B}_{\mathcal{P}}$ is the set of all ground atoms constructed from the predicates appearing in \mathcal{P} and the constants from $U_{\mathcal{P}}$. A term (resp. an atom, a literal, a rule or a program) is *ground* if no variable occur in it. A rule r' is a *ground instance* of a rule r if r' is obtained from r by replacing every variable in r with some constant in $U_{\mathcal{P}}$; $\text{ground}(\mathcal{P})$ denotes the set of all ground instances of the rules in \mathcal{P} . An interpretation M for a disjunctive program \mathcal{P} is any subset of $\mathcal{B}_{\mathcal{P}}$; M is a model of \mathcal{P} if it satisfies all rules in $\text{ground}(\mathcal{P})$. The (model-theoretic) semantics for positive \mathcal{P} assigns to \mathcal{P} the set of its *minimal models* $\mathcal{MM}(\mathcal{P})$, where a model M for \mathcal{P} is minimal if no proper subset of M is a model for \mathcal{P} . For any interpretation

¹ The order of literals in a conjunction or in a disjunction is immaterial. A literal can appear in a conjunction or in a disjunction at most once. The meaning of the symbols ‘ \wedge ’ and ‘ $,$ ’ is the same.

I , \mathcal{P}^I is the ground positive program derived from $ground(\mathcal{P})$ by 1) removing all rules that contain a negative literal $not a$ in the body and $a \in I$, and 2) removing all negative literals from the remaining rules. An interpretation M is a (*disjunctive*) *stable model* of \mathcal{P} if and only if $M \in \mathcal{MM}(\mathcal{P}^M)$. For general \mathcal{P} , the stable model semantics assigns to \mathcal{P} the set $\mathcal{SM}(\mathcal{P})$ of its stable models. It is well known that stable models are minimal models (i.e. $\mathcal{SM}(\mathcal{P}) \subseteq \mathcal{MM}(\mathcal{P})$) and that for negation free programs minimal and stable model semantics coincide (i.e. $\mathcal{SM}(\mathcal{P}) = \mathcal{MM}(\mathcal{P})$).

2.1 Queries

Predicate symbols are partitioned into two distinct sets: *base predicates* and *derived predicates*. Base predicates correspond to database relations defined over a given domain and they do not appear in the head of any rule whereas derived predicates are defined by means of rules. Given a set of ground atoms \mathcal{DB} , a predicate symbol r and a program \mathcal{P} , $\mathcal{DB}[r]$ denotes the set of r -tuples in \mathcal{DB} . The semantics of $\mathcal{P} \cup \mathcal{DB}$ is given by the set of its stable models by considering either their union (*possible semantics* or *brave reasoning*) or their intersection (*certain semantics* or *cautious reasoning*). A *query* Q is a pair (g, \mathcal{P}) where g is a predicate symbol, called the *query goal*, and \mathcal{P} is a program. The answer to a query $Q = (g, \mathcal{P})$ over a database \mathcal{DB} , under the possible (resp. certain) semantics is given by $\mathcal{DB}'[g]$ where $\mathcal{DB}' = \bigcup_{M \in \mathcal{SM}(\mathcal{P} \cup \mathcal{DB})} M$ (resp. $\mathcal{DB}' = \bigcap_{M \in \mathcal{SM}(\mathcal{P} \cup \mathcal{DB})} M$).

A (relational) query can be expressed by means of ‘safe’ non recursive Datalog, even though alternative equivalent languages such as relational algebra could be used as well [11, 21]. In the following queries are assumed to be expressed by means of stratified Datalog.

3 Databases and Integrity Constraints

Database schemata contain knowledge on the structure of data, i.e. they give constraints on the form the data must have. The relationships among data are usually defined by constraints such as functional dependencies, inclusion dependencies and others. Integrity constraints, which express information that is not directly derivable from the database, are introduced to prevent the insertion or deletion of data which could produce incorrect states. They are used to restrict the state a database can take and provide information on the relationships among data. Generally, a database \mathcal{DB} has an associated schema $\langle \mathcal{DS}, \mathcal{IC} \rangle$ defining the intentional properties of \mathcal{DB} : \mathcal{DS} denotes the structure of the relations, while \mathcal{IC} denotes the set of integrity constraints expressing semantic information over data.

3.1 Integrity Constraints

Definition 1. An *integrity constraint* is a formula of the first order predicate calculus of the form:

$$(\forall X)[\bigwedge_{j=1}^m b_j(X_j), \varphi(X_0) \supset \bigvee_{j=m+1}^n (\exists Z_j) b_j(X_j, Z_j)]$$

where $\varphi(X_0)$ denotes a conjunction of built-in atoms, $X = \bigcup_{j=1}^m X_j$, $X_i \subseteq X$ for $i \in [0..n]$ and all existentially quantified variables appear once. \square

Constraints will often be written in a different format by moving literals from the head to the body and vice versa. For instance, by rewriting the above constraint as denial the following one is obtained:

$$(\forall X) \left[\bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n (\exists Z_j) b_j(X_j, Z_j), \varphi(X_0) \supset \right].$$

Constraints without existential variables are called *full* or *universally quantified*. The reason for considering constraints of the above form is that we want to consider range restricted formulae, i.e. constraints whose variables either take values from finite domains only or the exact knowledge of their values is not relevant [21]. In the following the set of integrity constraints \mathcal{IC} is assumed to be satisfiable, that is there exists a database instance \mathcal{DB} satisfying \mathcal{IC} . For instance, by considering constraints of the above form with $m > 0$, the constraints are satisfied by the empty database.

3.2 Repairing and Querying Inconsistent Databases

An update atom is of the form $+a(X)$ or $-a(X)$, where $a(X)$ is a standard atom. A ground atom $+a(t)$ states that $a(t)$ will be inserted into the database, whereas a ground atom $-a(t)$ states that $a(t)$ will be deleted from the database. Given an update atom $+a(X)$ (resp. $-a(X)$), $Comp(+a(X)) = \text{not } a(X)$ (resp. $Comp(-a(X)) = a(X)$) denotes the complementary literal of the update atom $+a(X)$ (resp. $-a(X)$). Given a set \mathcal{U} of ground update atoms, the following sets are defined: $\mathcal{U}^+ = \{a(t) \mid +a(t) \in \mathcal{U}\}$, $\mathcal{U}^- = \{a(t) \mid -a(t) \in \mathcal{U}\}$ and $Comp(\mathcal{U}) = \{Comp(\pm a(t)) \mid \pm a(t) \in \mathcal{U}\}$. \mathcal{U} is said to be *consistent* if it does not contain two update atom $+a(t)$ and $-a(t)$ (i.e. if $\mathcal{U}^+ \cap \mathcal{U}^- = \emptyset$). Given a database \mathcal{DB} and a consistent set of update atoms \mathcal{U} , $\mathcal{U}(\mathcal{DB})$ denotes the updated database $\mathcal{DB} \cup \mathcal{U}^+ - \mathcal{U}^-$.

Definition 2. Given a database \mathcal{DB} and a set of integrity constraints \mathcal{IC} , a *repair* for $\langle \mathcal{DB}, \mathcal{IC} \rangle$ is a consistent set of update atoms \mathcal{R} such that 1) $\mathcal{R}(\mathcal{DB}) \models \mathcal{IC}$ and 2) there is no consistent set of update atoms $\mathcal{U} \subset \mathcal{R}$ such that $\mathcal{U}(\mathcal{DB}) \models \mathcal{IC}$. \square

Repaired databases are consistent databases, derived from the source database by means of a minimal set of update operations. Given a database \mathcal{DB} and a set of integrity constraints \mathcal{IC} , the set of all possible repairs for $\langle \mathcal{DB}, \mathcal{IC} \rangle$ is denoted as $\mathbf{R}(\mathcal{DB}, \mathcal{IC})$.

Definition 3. Given a database \mathcal{DB} and a set of integrity constraints \mathcal{IC} , an atom A is *true* (resp. *false*) with respect to \mathcal{IC} if A belongs to all repaired databases (resp. there is no repaired database containing A). The atoms which are neither true nor false are *undefined*. \square

Thus, true atoms appear in all repaired databases, whereas undefined atoms appear in a non empty proper subset of repaired databases. Given a database

\mathcal{DB} and a set of integrity constraints \mathcal{IC} , the application of \mathcal{IC} to \mathcal{DB} , denoted by $\mathcal{IC}(\mathcal{DB})$, defines three distinct sets of atoms: the set of true atoms $\mathcal{IC}(\mathcal{DB})^+$, the set of undefined atoms $\mathcal{IC}(\mathcal{DB})^u$ and the set of false atoms $\mathcal{IC}(\mathcal{DB})^-$.

Definition 4. Given a database \mathcal{DB} , a set of integrity constraints \mathcal{IC} and a query $Q = (g, \mathcal{P})$, the *consistent answer* of the query Q on the database \mathcal{DB} , denoted as $Q(\mathcal{DB}, \mathcal{IC})$, gives three sets, denoted as $Q(\mathcal{DB}, \mathcal{IC})^+$, $Q(\mathcal{DB}, \mathcal{IC})^-$ and $Q(\mathcal{DB}, \mathcal{IC})^u$. These contain, respectively, the sets of g -tuples which are *true* (i.e. belonging to $\bigcap_{R \in \mathbf{R}(\mathcal{DB}, \mathcal{IC})} Q(R(\mathcal{DB}))$), *false* (i.e. not belonging to $\bigcup_{R \in \mathbf{R}(\mathcal{DB}, \mathcal{IC})} Q(R(\mathcal{DB}))$) and *undefined* (i.e. set of tuples which are neither true nor false). \square

Next it is shown that repairs can be produced by logic programs derived from rules defining integrity constraints. Anyhow, as the presence of existentially quantified variables could produce a possibly infinite number of repairs, only restricted constraints from which we derive *safe* logic programs are allowed [21]. In more details it is allowed to express only constraints producing *domain independent* repairs, i.e. repairs whose atoms are constructed by only using database values [21].

4 Prioritized Active Integrity Constraints

In this section an extension of integrity constraints, which allows us to specify for each constraint the actions to be performed to satisfy it and preferences between them, is presented. The actions are defined by means of insertions and deletions.

Definition 5. A (universally quantified) *Active Integrity Constraint* (*AIC*) is of the form:

$$(\forall X) \left[\bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n \text{not } b_j(X_j), \varphi(X_0) \supset \bigvee_{i=1}^p \pm a_i(Y_i) \right] \quad (1)$$

where $X = \bigcup_{j=1}^m X_j$, $X_i \subseteq X$ for $i \in [0..n]$ and $Y_i \subseteq X$ for $i \in [1..p]$. \square

In the above definition the conditions $X = \bigcup_{j=1}^m X_j$, $X_i \subseteq X$ for $i \in [0..n]$ and $Y_i \subseteq X$ for $i \in [1..p]$ guarantee that variables are range restricted.

Active integrity constraints contain in the head the actions to be performed if the constraint defined in the body is not satisfied.

Given an AIC r of the form (1) $St(r)$ denotes the standard constraint:

$$(\forall X) \left[\bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n \text{not } b_j(X_j), \varphi(X_0) \supset \right]$$

derived from r by removing the head update atoms. Moreover, for a set of active integrity constraints \mathcal{IC} , $St(\mathcal{IC})$ denotes the corresponding set of standard integrity constraints, i.e. $St(\mathcal{IC}) = \{St(r) \mid r \in \mathcal{IC}\}$.

Definition 6. A (universally quantified) *Prioritized Active Integrity Constraint* (*PAIC*) is of the form:

$$(\forall X) \left[\bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n \text{not } b_j(X_j), \varphi(X_0) \supset \bigvee_{i=1}^{p_1} \pm a_i^1(Y_i^1) \succ \cdots \succ \bigvee_{i=1}^{p_k} \pm a_i^k(Y_i^k) \right] \quad (2)$$

where $X = \bigcup_{j=1}^m X_j$, $X_i \subseteq X$ for $i \in [0..n]$ and $Y_i^j \subseteq X$ for $j \in [1..k]$ and $i \in [1..p_j]$. \square

Prioritized active integrity constraints contain in the head the actions to be performed if the constraint defined in the body is not satisfied and express preferences among them.

Intuitively, the meaning of $\bigvee_{i=1}^{p_1} \pm a_i^1(Y_i^1) \succ \bigvee_{i=1}^{p_2} \pm a_i^2(Y_i^2)$ is that the actions $\pm a_1^1(Y_1^1), \dots, \pm a_{p_1}^1(Y_1^1)$ are preferable to the actions $\pm a_1^2(Y_1^2), \dots, \pm a_{p_2}^2(Y_1^2)$.

Given a (P)AIC $r = (\forall X)[\Phi \supset \Psi]$, Φ is called body of r (denoted by $\text{Body}(r)$), whereas Ψ is called head of r (denoted by $\text{Head}(r)$).

Definition 7. A (prioritized) active integrity constraint is said to be in *canonical* form if for each update literal $\pm a(X)$ appearing in the head, a literal $\text{Comp}(\pm a(X))$ also appears in the body. A set of (prioritized) active integrity constraints is said to be canonical if all constraints are in canonical form. \square

In the rest of the paper, (universally quantified) prioritized active integrity constraints in canonical form are considered. The motivation for restricting our attention to canonical AICs is due to the fact that in [6] it has been shown that for every ground AIC r , every head update atom $\pm A$ such that $\text{Comp}(\pm A) \notin \text{Body}(r)$ is useless and can be deleted.

Semantics

In the following firstly the definition concerning the truth value of ground atoms and ground update atoms with respect to a database \mathcal{DB} and a consistent set of update atoms \mathcal{U} is given, then the formal definition of founded and preferred repair is provided.

Definition 8. Given a database \mathcal{DB} and a consistent set of update atoms \mathcal{U} , the truth value of

- a positive ground literal $a(t)$ is *true* w.r.t. $(\mathcal{DB}, \mathcal{U})$ if $a(t) \in \mathcal{U}(\mathcal{DB})$,
- a negative ground literal $\text{not } a(t)$ is *true* w.r.t. $(\mathcal{DB}, \mathcal{U})$ if $a(t) \notin \mathcal{U}(\mathcal{DB})$,
- a ground update atom $\pm a(t)$ is *true* w.r.t. $(\mathcal{DB}, \mathcal{U})$ if $\pm a(t) \in \mathcal{U}$,
- built-in atoms, conjunctions and disjunctions of literals is given in the standard way,
- a ground AIC $\phi \supset \psi$ is *true* w.r.t. $(\mathcal{DB}, \mathcal{U})$ if ϕ is *false* w.r.t. $(\mathcal{DB}, \mathcal{U})$. \square

Definition 9. Let \mathcal{DB} be a database, \mathcal{IC} a set of AICs and \mathcal{R} a repair for $(\mathcal{DB}, \mathcal{IC})$.

- A ground update atom $\pm a(t) \in \mathcal{R}$ is *founded* if there exists $r \in \text{ground}(\mathcal{IC})$ s.t. $\pm a(t)$ appears in $\text{Head}(r)$ and $\text{Body}(r)$ is *true* w.r.t. $(\mathcal{DB}, \mathcal{R} - \{\pm a(t)\})$. We say that $\pm a(t)$ is *supported* by r w.r.t. \mathcal{R} .
- A ground rule $r \in \text{ground}(\mathcal{IC})$ is *applied* w.r.t. $(\mathcal{DB}, \mathcal{R})$ if there exists $\pm a(t) \in \mathcal{R}$ s.t. $\pm a(t)$ appears in $\text{Head}(r)$ and $\text{Body}(r)$ is *true* w.r.t. $(\mathcal{DB}, \mathcal{R} - \{\pm a(t)\})$. We say that r *supports* $\pm a(t)$ w.r.t. \mathcal{R} .

- \mathcal{R} is *founded* if all its atoms are *founded*.
- \mathcal{R} is *unfounded* if it is not founded.

□

The set of founded update atoms in \mathcal{R} with respect to $\langle \mathcal{DB}, \mathcal{IC} \rangle$ is denoted as $Founded(\mathcal{R}, \mathcal{DB}, \mathcal{IC})$, whereas $Unfounded(\mathcal{R}, \mathcal{DB}, \mathcal{IC}) = \mathcal{R} - Founded(\mathcal{R}, \mathcal{DB}, \mathcal{IC})$. Thus, update atoms of founded repairs are inferable by means of AICs. Given a database \mathcal{DB} and a set of AICs \mathcal{IC} , $\mathbf{FR}(\mathcal{DB}, \mathcal{IC})$ (resp. $\mathbf{R}(\mathcal{DB}, \mathcal{IC})$) denotes the set of founded repairs (resp. all the repairs) for $\langle \mathcal{DB}, \mathcal{IC} \rangle$. Clearly, the set of founded repairs is contained in the set of repairs ($\mathbf{FR}(\mathcal{DB}, \mathcal{IC}) \subseteq \mathbf{R}(\mathcal{DB}, St(\mathcal{IC}))$).

Example 3. Consider the following set of AICs \mathcal{IC} :

$$\forall(E, P, D)[mgr(E, P), prj(P, D), not\ emp(E, D) \supset +emp(E, D)]$$

$$\forall(E, D_1, D_2)[emp(E, D_1), emp(E, D_2), D_1 \neq D_2 \supset -emp(E, D_1) \vee -emp(E, D_2)]$$

The first constraint states that every manager E of a project P carried out by a department D must be an employee of D , whereas the second one says that every employee must be in only one department. Consider now the database $\mathcal{DB} = \{ mgr(e_1, p_1), prj(p_1, d_1), emp(e_1, d_2) \}$. There are three repairs for \mathcal{DB} : $\mathcal{R}_1 = \{-mgr(e_1, p_1)\}$, $\mathcal{R}_2 = \{-prj(p_1, d_1)\}$ and $\mathcal{R}_3 = \{+emp(e_1, d_1), -emp(e_1, d_2)\}$. \mathcal{R}_3 is the only founded repair as only the update atoms $+emp(e_1, d_1)$ and $-emp(e_1, d_2)$ are derivable from \mathcal{IC} . □

Definition 10. Let c be a PAIC and \mathcal{IC} a set of PAICs, then

- $\mathcal{AC}(c)$ denotes the active constraint derived from c by replacing symbol \succ with \vee . Moreover, $\mathcal{AC}(\mathcal{IC}) = \{\mathcal{AC}(c) \mid c \in \mathcal{IC}\}$.
- $\mathcal{SC}(c)$ denotes the standard constraint derived from c by deleting the update atoms appearing in the head. Moreover, $\mathcal{SC}(\mathcal{IC}) = \{\mathcal{SC}(c) \mid c \in \mathcal{IC}\}$ (i.e. $\mathcal{SC}(\mathcal{IC}) = St(\mathcal{AC}(\mathcal{IC}))$).
- $\mathcal{CC}(c)$ denotes the active constraint derived from $\mathcal{SC}(c)$ by inserting an update atom $\pm a(X)$ in the head if $Comp(\pm a(X))$ appears in the body of c . Moreover, $\mathcal{CC}(\mathcal{IC}) = \{\mathcal{CC}(c) \mid c \in \mathcal{IC}\}$. □

Example 4. Consider the following set of prioritized active integrity constraints \mathcal{IC} :

$$\begin{aligned} c, not\ a, not\ b &\supset +a \succ +b \succ -c \\ c, not\ d &\supset -c \end{aligned}$$

The following constraints can be derived:

- $\mathcal{AC}(\mathcal{IC})$ consists of the active constraints

$$\begin{aligned} c, not\ a, not\ b &\supset +a \vee +b \vee -c \\ c, not\ d &\supset -c \end{aligned}$$
- $\mathcal{SC}(\mathcal{IC})$ consists of the standard constraints

$$\begin{aligned} c, not\ a, not\ b &\supset \\ c, not\ d &\supset \end{aligned}$$
- $\mathcal{CC}(\mathcal{IC})$ consists of the active constraints

$$\begin{aligned} c, not\ a, not\ b &\supset +a \vee +b \vee -c \\ c, not\ d &\supset -c \vee +d \end{aligned}$$

□

Given a database \mathcal{DB} and a set of PAICs \mathcal{IC} , the set of repairs (resp. founded repairs) for $\langle \mathcal{DB}, \mathcal{IC} \rangle$ is denoted by $\mathbf{R}(\mathcal{DB}, \mathcal{IC})$ (resp. $\mathbf{FR}(\mathcal{DB}, \mathcal{IC})$).

Fact 1. Given a database \mathcal{DB} and a set of PAICs \mathcal{IC}

- $\mathbf{R}(\mathcal{DB}, \mathcal{IC}) = \mathbf{R}(\mathcal{DB}, \mathcal{AC}(\mathcal{IC})) = \mathbf{R}(\mathcal{DB}, \mathcal{SC}(\mathcal{IC}))$
- $\mathbf{FR}(\mathcal{DB}, \mathcal{IC}) = \mathbf{FR}(\mathcal{DB}, \mathcal{AC}(\mathcal{IC}))$

□

The above fact states that the repairs for a database \mathcal{DB} and a set of PAICs \mathcal{IC} can be derived by considering the corresponding active (resp. standard) integrity constraints $\mathcal{AC}(\mathcal{IC})$ (resp. $\mathcal{SC}(\mathcal{IC})$), whereas founded repairs can be derived by considering active constraints $\mathcal{AC}(\mathcal{IC})$, obtained by replacing symbol \succ with \vee in the head of prioritized active integrity constraints.

Definition 11 (Preferences between repairs). Let \mathcal{DB} be a database and \mathcal{IC} a set of PAICs. For any repairs R_1 , R_2 and R_3 in $\mathbf{R}(\mathcal{DB}, \mathcal{IC})$, we say that:

- $R_1 \sqsupseteq R_1$.
- $R_1 \sqsupseteq R_2$ if:
 1. $R_1 \in \mathbf{FR}(\mathcal{DB}, \mathcal{IC})$ and $R_2 \notin \mathbf{FR}(\mathcal{DB}, \mathcal{IC})$, or
 2. (a) $R_1, R_2 \in \mathbf{FR}(\mathcal{DB}, \mathcal{IC})$ or $R_1, R_2 \notin \mathbf{FR}(\mathcal{DB}, \mathcal{IC})$ and
 - (b) there are two update atoms $\pm a(t) \in R_1$ and $\pm b(u) \in R_2$ and a (ground) prioritized active integrity constraint c such that
 - (i) $\text{head}(c) = \dots \pm a(t) \dots \succ \dots \pm b(u) \dots$ and
 - (ii) c supports $\pm a(t)$ w.r.t. R_1 and $\pm b(u)$ w.r.t. R_2 .
- If $R_1 \sqsupseteq R_2$ and $R_2 \sqsupseteq R_3$, then $R_1 \sqsupseteq R_3$.

If $R_1 \sqsupseteq R_2$, then R_1 is *preferable* to R_2 . Moreover, if $R_1 \sqsupseteq R_2$ and $R_2 \not\sqsupseteq R_1$, then $R_1 \sqsupset R_2$. A repair R is a *preferred* repair if there is no repair R' such that $R' \sqsubset R$. □

The set of preferred repairs for a database \mathcal{DB} and a set of prioritized active integrity constraints \mathcal{IC} is denoted by $\mathbf{PR}(\mathcal{DB}, \mathcal{IC})$.

Example 5. Consider the database $\mathcal{DB} = \{c\}$ and the set of PAICs \mathcal{IC} of Example 4. $R_1 = \{-c\}$, $R_2 = \{+a, +d\}$ and $R_3 = \{+b, +d\}$ are the three repairs for $\langle \mathcal{DB}, \mathcal{IC} \rangle$; their relation is: $R_1 \sqsupset R_2 \sqsupset R_3$ and the preferred repair is R_1 . □

The next theorem states the relation between preferred, founded and general repairs.

Theorem 2. Let \mathcal{DB} be a database and \mathcal{IC} a set of PAICs, then

$$\mathbf{PR}(\mathcal{DB}, \mathcal{IC}) \begin{cases} \subseteq \mathbf{FR}(\mathcal{DB}, \mathcal{IC}) & \text{if } \mathbf{FR}(\mathcal{DB}, \mathcal{IC}) \neq \emptyset \\ \subseteq \mathbf{R}(\mathcal{DB}, \mathcal{IC}) & \text{if } \mathbf{FR}(\mathcal{DB}, \mathcal{IC}) = \emptyset \end{cases}$$

□

Given a database \mathcal{DB} , a set of prioritized integrity constraints \mathcal{IC} and a query $Q = (g, \mathcal{P})$, the *preferred consistent answer* of the query Q on the database \mathcal{DB} , denoted as $Q(\mathcal{DB}, \mathcal{IC})$, gives three sets, denoted as $Q(\mathcal{DB}, \mathcal{IC})^+$, $Q(\mathcal{DB}, \mathcal{IC})^-$ and $Q(\mathcal{DB}, \mathcal{IC})^u$. These contain, respectively, the sets of g -tuples which are *true* (i.e. belonging to $\bigcap_{R \in \mathbf{PR}(\mathcal{DB}, \mathcal{IC})} Q(R(\mathcal{DB}))$), *false* (i.e. not belonging to $\bigcup_{R \in \mathbf{PR}(\mathcal{DB}, \mathcal{IC})} Q(R(\mathcal{DB}))$) and *undefined* (i.e. set of tuples which are neither true nor false). It is worth noting that the preferred consistent answer of a query considers only the preferred repairs rather than all the repairs.

Desirable properties. We now introduce desirable properties on the set of preferred repairs. Properties which should be satisfied by families of preferred repairs have been introduced in [9]. Here we adapt properties defined in [9], where preferences are static, to our framework.

Given a database \mathcal{DB} and a set of PAICs \mathcal{IC} , then the following properties can be identified:

- **Non-emptiness:** $\langle \mathcal{DB}, \mathcal{IC} \rangle$ always admits some preferred repairs.
- **Monotonicity:** adding preference information can only narrow the set of preferred repairs.
- **Non-discrimination:** if no preference information is expressed, any repair is a preferred repair.

Given two sets of PAICs \mathcal{IC}_1 and \mathcal{IC}_2 , we say that $\mathcal{IC}_1 \triangleleft \mathcal{IC}_2$ if \mathcal{IC}_1 is derived from \mathcal{IC}_2 by replacing one or more \succ symbols with the \vee symbol.

Definition 12. Given a set of PAICs \mathcal{IC} and two relation symbols a and b , we say that $\pm a$ (inserting into or deleting from a) is preferable to $\pm b$ (inserting into or deleting from b) w.r.t. \mathcal{IC} and we write $\pm a \gg_{\mathcal{IC}} \pm b$, if:

1. there is a PAIC $c \in \mathcal{IC}$ such that $Head(c) = \dots \pm a(X) \dots \succ \dots \pm b(Y) \dots$, or
2. there exists $\pm c$ such that $\pm a \gg_{\mathcal{IC}} \pm c$ and $\pm c \gg_{\mathcal{IC}} \pm b$. \square

Observe that the above condition could be relaxed by considering ground atoms instead of predicate symbols.

The following theorem shows some properties of PAICs.

Theorem 3. Let \mathcal{DB} be a database and \mathcal{IC} a set of PAICs.

1. **Non-emptiness:** $\mathbf{PR}(\mathcal{DB}, \mathcal{IC}) \neq \emptyset$ if \mathcal{IC} is satisfiable.
2. **Monotonicity:** $\mathcal{IC}' \triangleleft \mathcal{IC} \Rightarrow \mathbf{PR}(\mathcal{DB}, \mathcal{IC}) \subseteq \mathbf{PR}(\mathcal{DB}, \mathcal{IC}')$ if $\gg_{\mathcal{IC}}$ is acyclic, i.e. there does not exist an update atom $\pm a$ such that $\pm a \gg_{\mathcal{IC}} \pm a$. \square
3. **Non-discrimination:** if $\mathcal{IC} = \mathcal{CC}(\mathcal{IC})$ then $\mathbf{PR}(\mathcal{DB}, \mathcal{IC}) = \mathbf{R}(\mathcal{DB}, \mathcal{IC})$. \square

4.1 Computing Repairs Through Datalog Programs

A general approach for the computation of repairs and consistent answers in the presence of databases with universal integrity constraints has been proposed in [16]. The technique is based on the generation of a disjunctive program $\mathcal{DP}(\mathcal{IC})$ derived from the set of integrity constraints \mathcal{IC} . The repairs for a database \mathcal{DB} can be generated from the stable models of $\mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB}$. We now present an extension of this technique for prioritized active integrity constraints in canonical form.

Definition 13. Given a set of prioritized active integrity constraints \mathcal{IC} , then $\mathcal{DP}(\mathcal{IC})$ is the disjunctive datalog program derived from \mathcal{IC} (or equivalently from $\mathcal{SC}(\mathcal{IC})$) by replacing a PAIC of the form (2) with a disjunctive rule of the form:

² Note that also the approach proposed in [9] satisfies monotonicity property under the assumption that the priority relation is acyclic.

$$\bigvee_{j=1}^m \neg b_j(X_j) \vee \bigvee_{j=m+1}^n +b_j(X_j) \leftarrow \bigwedge_{j=1}^m (b_j(X_j) \vee +b_j(X_j)), \bigwedge_{j=m+1}^n (\text{not } b_j(X_j) \vee \neg b_j(X_j)), \varphi(X_0)$$

and by adding a constraint

$$\leftarrow \neg b(X), +b(X)$$

for each predicate symbol b . \square

Example 6. Consider the set of integrity constraints \mathcal{IC} of Example 4. The following set of rules $\mathcal{DP}(\mathcal{IC})$ can be derived:

$$\begin{aligned} +a \vee +b \vee -c &\leftarrow (c \vee +c), (\text{not } a \vee -a), (\text{not } b \vee -b) \\ -c \vee +d &\leftarrow (c \vee +c), (\text{not } d \vee -d) \\ &\leftarrow +a, -a \\ &\leftarrow +b, -b \\ &\leftarrow +c, -c \\ &\leftarrow +d, -d \end{aligned}$$

\square

It is worth noting that, in considering atoms of the form $+a(t)$, $-a(t)$ and $a(t)$, the symbols $+a$, $-a$ and a are assumed to be different predicate symbols.

Definition 14. Given an interpretation M , we denote as $\text{UpdateAtoms}(M)$ the set of update atoms in M . Given a set S of interpretations, we define $\text{UpdateAtoms}(S) = \{\text{UpdateAtoms}(M) | M \in S\}$. \square

Theorem 4. Given a database \mathcal{DB} and a set of canonical prioritized active integrity constraints \mathcal{IC} , then:

- (Soundness) for every stable model M of $\mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB}$, $\text{UpdateAtoms}(M)$ is a repair for $\langle \mathcal{DB}, \mathcal{IC} \rangle$;
- (Completeness) for every database repair S for $\langle \mathcal{DB}, \mathcal{IC} \rangle$ there exists a stable model M of $\mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB}$ such that $S = \text{UpdateAtoms}(M)$. \square

Example 7. Consider the database $\mathcal{DB} = \{c\}$ and the set of PAICs \mathcal{IC} of Example 4. The stable models of $\mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB}$ are $M_1 = \{c, \neg c\}$, $M_2 = \{c, +a, +d\}$ and $M_3 = \{c, +b, +d\}$. Each stable model corresponds to a repair for $\langle \mathcal{DB}, \mathcal{IC} \rangle$ and vice versa. \square

Definition 15. Given a set of prioritized active integrity constraint \mathcal{IC} , then $\mathcal{FP}(\mathcal{IC})$ is the datalog program obtained as follows:

- for each update atom $\pm a(X)$ defined in $\mathcal{DP}(\mathcal{IC})$, the following rule is introduced:

$$\leftarrow \pm a(X), \text{not founded} \pm a(X)$$

- if $\pm a(X)$ appears in the head of a constraint $c \in \mathcal{AC}(\mathcal{IC})$, that is c is of the form:

$$(\forall X)[\bigwedge_{j=1}^m b_j(X_j), \bigwedge_{j=m+1}^n \text{not } b_j(X_j), \text{Comp}(\pm a(X)), \varphi(X_0) \supset \bigvee_{i=1}^p \pm a_i(Y_i) \vee \pm a(X)]$$

then the following rule is introduced:

$$\begin{aligned} \text{founded} \pm a(X) &\leftarrow \text{Comp}(\pm a(X)), \bigwedge_{j=1}^m ((b_j(X_j) \wedge \text{not } \neg b_j(X_j)) \vee +b_j(X_j)), \\ &\quad \bigwedge_{j=m+1}^n (\text{not } b_j(X_j) \wedge \text{not } +b_j(X_j)) \vee \neg b_j(X_j), \varphi(X_0) \end{aligned}$$

Given a set of prioritized active integrity constraint \mathcal{IC} , then $\mathcal{FDP}(\mathcal{IC}) = \mathcal{DP}(\mathcal{IC}) \cup \mathcal{FP}(\mathcal{IC})$. \square

Example 8. Consider the set of integrity constraints \mathcal{IC} of Example 4. The following set of rules $\mathcal{FP}(\mathcal{IC})$ can be derived:

$$\begin{aligned} & \leftarrow +a, \text{not founded}+a \\ & \leftarrow +b, \text{not founded}+b \\ & \leftarrow -c, \text{not founded}-c \\ & \leftarrow +d, \text{not founded}+d \\ & \text{founded}+a \leftarrow \text{not } a, ((c \wedge \text{not } -c) \vee +c), ((\text{not } b \wedge \text{not } +b) \vee -b) \\ & \text{founded}+b \leftarrow \text{not } b, ((c \wedge \text{not } -c) \vee +c), ((\text{not } a \wedge \text{not } +a) \vee -a) \\ & \text{founded}-c \leftarrow c, ((\text{not } b \wedge \text{not } +b) \vee -b), ((\text{not } a \wedge \text{not } +a) \vee -a) \\ & \text{founded}-c \leftarrow c, ((\text{not } d \wedge \text{not } +d) \vee -d) \end{aligned}$$

 \square

Theorem 5. Given a database \mathcal{DB} and a set of canonical prioritized active integrity constraints \mathcal{IC} , then:

- (Soundness) for every stable model M of $\mathcal{FDP}(\mathcal{IC}) \cup \mathcal{DB}$, $\text{UpdateAtoms}(M)$ is a founded repair for $\langle \mathcal{DB}, \mathcal{IC} \rangle$;
- (Completeness) for every founded repair S for $\langle \mathcal{DB}, \mathcal{IC} \rangle$ there exists a stable model M of $\mathcal{FDP}(\mathcal{IC}) \cup \mathcal{DB}$ such that $S = \text{UpdateAtoms}(M)$. \square

Example 9. Consider the database $\mathcal{DB} = \{c\}$ and the set of PAICs \mathcal{IC} of Example 4. The unique stable model of $\mathcal{FDP}(\mathcal{IC}) \cup \mathcal{DB}$ is $M = \{c, -c\}$ whose update atoms correspond to the unique founded repair for $\langle \mathcal{DB}, \mathcal{IC} \rangle$. \square

Observe that given a set of PAICs \mathcal{IC} and a database \mathcal{DB} , then $\mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB}$ gives all the possible repairs for $\langle \mathcal{DB}, \mathcal{IC} \rangle$, whereas $\mathcal{DP}(\mathcal{IC}) \cup \mathcal{FP}(\mathcal{IC}) \cup \mathcal{DB}$ gives only the founded repairs as the constraints in $\mathcal{FP}(\mathcal{IC})$ discard every stable model of $\mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB}$ which does not correspond to a founded repair.

Theorem 6. Let \mathcal{DB} be a database and \mathcal{IC} a set of prioritized active integrity constraints, then

$$\mathbf{PR}(\mathcal{DB}, \mathcal{IC}) \begin{cases} \subseteq \text{UpdateAtoms}(\mathcal{SM}(\mathcal{FDP}(\mathcal{IC}) \cup \mathcal{DB})) \text{ if } \mathcal{SM}(\mathcal{FDP}(\mathcal{IC}) \cup \mathcal{DB}) \neq \emptyset \\ \subseteq \text{UpdateAtoms}(\mathcal{SM}(\mathcal{DP}(\mathcal{IC}) \cup \mathcal{DB})) \quad \text{if } \mathcal{SM}(\mathcal{FDP}(\mathcal{IC}) \cup \mathcal{DB}) = \emptyset \end{cases} \quad \square$$

5 Conclusions

This paper has introduced *prioritized active integrity constraints*, a simple and powerful form of active rules with declarative semantics, well suited for computing (preferred) database repairs and consistent answers. A prioritized active integrity constraint defines an integrity constraint, the actions which should be performed if the constraint is not satisfied and preferences among these actions. These preferences determine a partial order among feasible repairs, so that preferred repairs can be selected among all the possible repairs. It has been shown that prioritized active integrity constraints can be rewritten into disjunctive Datalog programs and that repairs can be computed through the computation of stable models.

References

1. Abiteboul S., Hull R., Vianu V. *Foundations of Databases*. Addison-Wesley, 1994.
2. Arenas, M., Bertossi, L., Chomicki, J., Consistent query Answers in inconsistent databases. *PODS*, pp. 68–79, 1999.
3. Arenas, M., Bertossi, L., Chomicki, J., Specifying and Querying Database repairs using Logic Programs with Exceptions. *FQAS*, pp. 27-41, 2000.
4. Arenas, M., Bertossi, L., Chomicki, J., Answer sets for consistent query answering in inconsistent databases. *TPLP* 3(4-5), 393-424, 2003.
5. Brewka, G., Niemela, I., Truszczyński, M., Answer Set Optimization. *IJCAI*, 867-872, 2003.
6. Caroprese, L., Greco, S., Sirangelo, C., and Zumpano, E., Declarative Semantics of Production Rules for Integrity Maintenance. *ICLP*, 26-40, 2006
7. Chomicki, J., Preference Formulas in Relational Queries. *ACM TODS*, 28(4), 1-40, 2003.
8. Chomicki, J., Lobo, J., Naqvi, S. A., Conflict Resolution Using Logic Programming. *IEEE Trans. Knowl. Data Eng.* 15(1), pp. 244-249, 2003.
9. Chomicki, J., Staworko, S., and Marcinkowski, J., Preference-Driven Querying of Inconsistent Relational Databases. *Proc. International Workshop on Inconsistency and Incompleteness in Databases*, 2006.
10. Delgrande, J., P., Schaub, T., Tompits, H., A Framework for Compiling Preferences in Logic Programs. *TPLP*, 3(2), 129-187, 2003.
11. Eiter, T., Gottlob, G., Mannila, H., Disjunctive Datalog. *TODS*, 22(3), 364–418, 1997.
12. Gelfond, M., Son, T.C., Reasoning with prioritized defaults. *LPKR*, 164-223, 1997.
13. Gelfond, M., Lifschitz, V. The Stable Model Semantics for Logic Programming, *ICLP*, 1988.
14. Gelfond, M., Lifschitz, V., Classical Negation in Logic Programs and Disjunctive Databases, *NGC*, No. 9, 365–385, 1991.
15. Grant, J., Subrahmanian, V. S., Reasoning in Inconsistent Knowledge Bases, *TKDE*,7(1), 177-189, 1995.
16. Greco S., and Zumpano E., Querying Inconsistent Databases. *LPAR*, 308–325, 2000.
17. Greco G., Greco S., and Zumpano E., A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE TKDE*, 15(6), 1389-1408, 2003.
18. Lin J., Mendelzon A. O., Merging Databases Under Constraints. *Int. J. Cooperative Inf. Syst.*, 7(1), 55-76, 1998.
19. Marek, V. W., Truszczyński, M., Revision Programming. *Theoretical Computer Science* 190(2), pp. 241-277, 1998.
20. Sakama, C., Inoue, K., Priorized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123, 185-222, 2000.
21. Ullman, J. K., *Principles of Database and Knowledge-Base Systems*, Vol. 1, Computer Science Press, 1988.
22. Wijsen, J., Condensed representation of database repairs for consistent query answering, *ICDT*, 378-393, 2003.
23. Zhang, Y., Foo, N., Answer sets for prioritized logic programs. *ILPS*, 69-83, 1997.

Using Redundant Bit Vectors for Near-Duplicate Image Detection

Jun Jie Foo and Ranjan Sinha

School of Computer Science & IT
RMIT University, Melbourne, Australia, 3001
{jufoo,rsinha}@cs.rmit.edu.au

Abstract. Images are amongst the most widely proliferated form of digital information due to affordable imaging technologies and the Web. In such an environment, the use of digital watermarking for image copyright infringement detection is a challenge. For such tasks, near-duplicate image detection is increasingly attractive due to its ability of automated content analysis; moreover, the application domain also extends to data management. The application of PCA-SIFT features and Locality-Sensitive Hashing (LSH) — for indexing and retrieval — has been shown to be highly effective for this task. In this work, we prune the number of PCA-SIFT features and introduce a modified Redundant Bit Vector (RBV) index. This is the first application of the RBV index that shows near-perfect effectiveness. Using the best parameters of our RBV approach, we observe an average *recall* and *precision* of 91% and 98%, respectively, with query response time of under 10 seconds on a collection of 20,000 images. Compared to the baseline (the LSH index), the query response times and index size of the RBV index is 12 times faster and 126 times smaller, respectively. As compared to brute-force sequential scan, the RBV index rapidly reduces the search space to 1/80.

Keywords: Near-duplicate Image Detection, Redundant Bit-Vectors, RBV.

1 Introduction

Many digital images available on the Web are copies or variants of each other; these include the scaled-down thumbnails kept by web search engines and differing versions of a single image made available by different news portals. Online images can be appropriated without the acknowledgment of source and, accidentally or otherwise, disguised through simple processing. Common modifications include conversion to greyscale, change in color balance, rescaling, rotating, cropping, and filtering operations. For reasons such as copyright infringement detection [14] and collection management [5], it is attractive to identify such variants (*near-duplicates*) with a reasonable degree of reliability.

In recent work, Qamra et al. [14] propose the Perceptual Distance Functions (PDF) for near-duplicate detection using color and texture image features. However, only mediocre effectiveness is observed when these functions are used with approximate indexing structures such as the Locality-Sensitive Hashing (LSH)

index [6]. To our knowledge, the highest reported accuracy of a near-duplicate image detection system is that of Ke et al. [11] — henceforth referred to as the KSH system — that uses the PCA-SIFT descriptors [10] (an extension of the SIFT interest points [12]) and the LSH index. Near-perfect precision is reported for retrieval of various altered images — cropped, scaled, rotated, and various affine transformation. While high precision has been reported, scalability has not been explored and the reported interactive query response times are observed on a tiny image collection. In this work, we use the KSH system as the baseline.

In this paper, we propose pruning strategies on the SIFT interest points (characterized by the PCA-SIFT local descriptors) along with the use of modified Redundant Bit Vectors (RBV) [8] for this application domain. While RBV has been reported to be efficient for negative queries on audio fingerprint detection, it has yet to be demonstrated to work for positive queries. Here, we propose a novel scheme that extends the RBV index for positive queries. Using our proposed scheme on pruned PCA-SIFT descriptors, we report an almost lossless level of effectiveness for this particular application. We also demonstrate that this index can be highly compact as compared to the LSH index (as used in the KSH system).

2 Distinctive Interest Points

Given an image, the idea of local descriptors is to detect image regions (centered around interest points) that possess properties invariant to geometric variation and photometric changes, so that distinctive local descriptors can be computed for each region [9][13].

In this work, we use the popular SIFT (scale invariant feature transform) detector [12] that has been demonstrated to outperform existing detectors [13]. We apply the PCA-SIFT descriptors on the SIFT interest points instead of the original SIFT descriptors, as they are shown to be both highly distinctive [10] and highly effective for near-duplicate image detection [11]. In this work, for convenience, the SIFT keypoint detector and the PCA-SIFT local descriptor are referred to as a PCA-SIFT feature.

There are four major stages of computation in the SIFT detector for extracting a set of image features, namely the scale-space extrema detection, *keypoint* localization, orientation assignment, and generation of keypoint local descriptor.

In the first phase of the SIFT keypoint detector, the difference-of-Gaussian (DoG) function is used to identify candidate points in various locations and scales using a Gaussian pyramid. This is achieved by finding local peaks (keypoints) in a series of DoG images. In the second phase, poorly localized and unstable keypoints — below a threshold level — are rejected. After all stable keypoints are identified, each keypoint is assigned a dominant orientation for rotation invariance in the third phase. Additional keypoints are generated if there are multiple orientations within 80% threshold of the dominant orientation; thus, there can be multiple keypoints with identical scale, location, but different orientation.

The PCA-SIFT descriptors are computed using the same information as the original SIFT descriptor, that is, location, scale, and dominant orientations. PCA-SIFT concatenates the horizontal and vertical gradient maps for the 41×41

region — centered around the keypoint, rotated to align its orientation to a canonical direction — to produce a $2 \times 39 \times 39 = 3042$ element local descriptor (feature vector).

In cases where there are multiple dominant orientations, a separate vector is calculated for each. Each vector is then projected — using principal component analysis, a common technique for dimensionality reduction — to a low-dimensional feature space using a pre-computed eigenspace¹. Ke et al. [11] have empirically determined that $n = 36$ feature spaces for the local descriptor performs well for near-duplicate image retrieval; wherein any two PCA-SIFT local descriptors are deemed similar (a match) within an Euclidean distance (L_2 -norm) of 3000. In this work, we use the same settings.

The problem of applying PCA-SIFT features is that each image consists of hundreds to thousands of high-dimensional local descriptors, and a reliable match between two images requires at least 3 to 5 descriptor matches [12]. The KSH index uses Locality Sensitive Hashing (LSH) [6] for indexing these PCA-SIFT features. We refer interested readers to the work of Ke et al. [11] for further discussion.

3 Keypoint Reduction

Querying in high-dimensional space is a challenging problem due to the *curse of dimensionality* [1]; indeed, this is further amplified as the evaluation of a query image using PCA-SIFT features requires multiple point matches in high-dimensional space — simulating multiple point queries.

We observe an average of 1,400 keypoints per image for our image collection, similar to the reported average of 1,100 in the work of Ke et al. [11]. In practice, using PCA-SIFT features, each image can generate from a few hundred up to a few thousand local descriptors of 36 dimensions each depending on the complexity of an image. Hence, the reduction of keypoints that SIFT generates per image is key to a scalable system. Given that the SIFT interest point detector was originally proposed as a distinctive feature for matching objects or image scenes with high variance [12], it is apparent that all keypoints are required for robust matching. We hypothesize that near-duplicate image detection requires only a subset of the keypoints as we only consider images that are derived from the same source, where the level of variance is limited.

In the second stage of the SIFT keypoint detector algorithm, Lowe [12] has empirically observed that a contrast threshold value of 0.03 — used to eliminate keypoints with low contrast — yields good results. This is an important parameter as a higher threshold will result in fewer keypoints being generated. To exploit this observation for our application, we select only the top N most significant keypoints by their contrast. By setting an upper bound on the number of keypoints that are selected in this phase, we immediately prune more than 80% (on average) of the keypoints required for each image. Images that do not have N detected keypoints are not pruned. Since some keypoints may share sub-pixel

¹ The eigenspace used in this work is provided by Ke et al. [10].

location and scale information with multiple orientation, we expect approximately 15% additional keypoints to be generated in the subsequent phase [12].

This approach has been demonstrated to achieve comparable effectiveness to using all keypoints for this application, while simultaneously reducing response times considerably during query evaluation [4].

4 Redundant Bit Vectors

Goldstein et al. [8] propose Redundant Bit Vectors (RBV) for high-dimensionality search for multimedia data. The algorithm consists of three key ideas: 1) approximate high-dimensional spherical regions by tightened hyper-rectangles, 2) partition the query space to promote redundancy in the index, and 3) represent each partition with an efficient bit vector.

The conventional nearest-neighbor matching problem is usually formulated as a point query over spheres of fixed or variable radius. The ϵ -range search can be applied to return all objects with distances within an ϵ threshold [1] if more than one object is required. The distances between two objects are commonly measured in some L_p metric. Goldstein et al. [7] formulate this as a *rectangle search problem*, where each point p in a d -dimensional space is replaced by the smallest hypercube c enclosing the hyper-sphere with center point p . With this approach, the data space is searched using approximated rectangular regions instead of spheres.

To create an RBV index, all data points are mapped onto data (hyper) rectangles, where each dimension of these rectangles are partitioned into m bins. The choice of m is determined by the number of disjoint intervals between the data rectangles [7]. Every dimension of the rectangle is projected onto its respective axis, where each partition is a bit vector that reflects the overlap test between the interval boundaries. Each bit in a spatial bit vector corresponds to a data rectangle within the collection. Each adjacent bit vector (within the same dimension) may have identical bits set to 1 if the data rectangle overlaps the interval boundaries, hence giving rise to redundant bit vectors. To search for approximated neighbors for a given point, a spatial bit vector for a given dimension is selected if the partition includes the query point — has its corresponding bit set to 1. The resultant spatial bit vector after bitwise ANDing all selected vectors from every dimension will return the data rectangles that contains the query point.

Goldstein et al. demonstrate that the RBV index excel at applications where there is a large fraction of negative queries. For such applications, they report significant gains in efficiency, and a reduction of memory requirement in comparison to LSH [8]. However, their approach is unsuitable for applications with a large fraction of positive queries given that accuracy is sacrificed considerably for efficiency gains; we propose a novel scheme that extends the current RBV index to cater for such applications.

5 Extending the RBV Index

Using a similar scheme as Ke et al. [11], all indexed images are stored in a file table (FT), and each PCA-SIFT feature is mapped using a keypoint table (KT)

where each entry is 92 bytes and contains the location, scale, orientation, and local descriptor of a keypoint. For every keypoint in a query image — or query keypoint — we approximate the potential matching keypoints using an index and verify the short-listed matched pairs using geometric verification (RANSAC) [3]. The key differences are the employed indexing technique and the amount of features used. All query keypoints are read into memory during query evaluation, whereas every matched keypoint is fetched from disk. All disk reads are linearized for efficiency.

Given a collection of images, we index only the 36 dimensional local descriptors (keypoints). Each keypoint can be represented by $k_i(x_1 \dots x_d), i = 1 \dots N$, where N is the total number of keypoints in the collection, and x_d is the coordinate of dimension d . During RBV index construction, each point k_i is mapped to a data rectangle using the smallest hypercube c that encloses the hyper-sphere centered on k_i with radius of ϵ — for ϵ -range search. For two PCA-SIFT keypoint descriptors to be deemed a match, an L_2 norm ranging from 2,200 to 3,000 (ϵ) yields high effectiveness [10][11]. Hence, each keypoint is converted into a hypercube, with a hypercube half-sidelength (HCS) of ϵ where $c = 2\epsilon$.

We use $m_i = (x_{i\text{range}})/\epsilon$ partitions to create the desired number of disjoint intervals that cover the entire axis of a single dimension, where $x_{i\text{range}} = x_{imax} - x_{imin}$ for dimension i . Thus, the choice of HCS is critical given that it determines the granularity of the partitioning scheme. To create the partitions, we first sort the boundaries of all data hypercube in a given dimension along its axis. Each partition is represented using a bit vector where each bit reflects the index position in KT. Following [8], we then select $m_i - 1$ dividers from the sorted hypercube boundary values and partition the dimension using the overlap test for each interval, where each bit in a bit vector reflects the predicate (1 or 0).

The collection of data hypercube (keypoints) can be represented efficiently, as each integer can store up to 32 or 64 keypoint IDs, depending on the system architecture (machine word size). Each bit vector is represented using an array of integers, where the bit vectors are constructed in memory and written to disk, one dimension at a time. Each bit vector is stored using an array of $N/32$ (4-byte) integers, where N is the total number of keypoints in our collection. We also store $m_i - 1$ (4-byte) dividers for the axes of each dimension for query evaluation. Thus, the size of the index for the entire image collection is approximately

$$\sum_{i=1}^D m_i \left(4 \times \frac{N}{32} + 4 \times (m_i - 1) \right) \text{bytes}$$

where D is the number of dimensions; in our application D is 36.

In the original RBV indexing scheme, for efficiency gains, Goldstein et al. [8] sort the data points in ascending order of the most selective dimension (smallest amount of overlap) prior to constructing the bit vectors. This is done to organize the RBV index such that the first dimension will have data hypercubes closely located along the axis of its dimension, resulting in tightly packed bits of 1's between the *low* and *high* range. Since the number of bitwise AND operations can be reduced to the number of integers between this range, the most selective dimension is used as the first queried dimension. The ordering of dimensions

in which to query is important since the first dimension always dictates the resultant list of matching keypoints — using the bitwise AND operation. Querying the most selective dimension first will short-list the number of potential matching keypoints rapidly. However, given that each dimension is a very coarse approximation of the distance in the hypercube space, retrieval accuracy suffers.

As our aim is to maximize the number of positive matches, the index is modified to be less restrictive for this application. In our proposed scheme, the most selective dimension is not pre-determined, and requires no prior sorting of the data points; consequently, we do not utilize the low and high range for bit vector processing.

A summary of the process for constructing the modified RBV index is as follows:

Require: Database of N D -dimensional local descriptors x in KT, $\epsilon = HCS$.

```

for  $i = 1$  to  $D$  do
    for  $k = 1$  to  $N$  do
        Calculate hypercube boundaries  $x_{ki} \pm \epsilon$ .
    end for
    Sort boundaries on  $i$ -axis; calculate  $m_i = (i_{max} - i_{min})/\epsilon$ .
    Select  $m_i - 1$  dividers from sorted boundaries.
end for
for  $i = 1$  to  $D$  do
    for  $k = 1$  to  $N$  do
        Create overlap tests to create  $m_i$  partitions (bit-vectors).
    end for
    Store  $m_i - 1$  dividers and  $m_i$  bit-vectors to disk.
end for
```

Querying the modified RBV index. Instead of querying with the most selective dimension during index construction, we determine the order of dimensions dynamically during query evaluation, thereby eliminating the need for pre-processing the data points. For each $x_i, i = 1\dots d$ of a query keypoint, we determine the *normalized distance to mean* using $|x_\mu - x_i|/x_\mu$. We sort the distances in ascending order, and use the sorted order of dimensions for query evaluation. Thus, the dimensions are dynamically selected to maximize the potential keypoint matches to the query coordinates. In this approach, the search space is not immediately pruned with the first queried dimension but is instead narrowed progressively by processing each subsequent dimension.

During query evaluation, the required partition for each dimension can be calculated in memory by using the $m - 1$ dividers of each dimension to determine which partitions to retrieve from disk. Given that each bit vector is bitwise ANDed one dimension at a time, and that the ordering of dimensions can be pre-processed, we can bulk-process the query keypoints simultaneously. This is achieved by using a temporary resultant bit vector in memory for each query keypoint. Hence the order in which the required partitions are read can also be sorted to allow sequential access to disk. Bulk-processing of keypoints in memory is enabled by keypoint reduction since the feature space of the query images are pruned, without which the memory requirement would be high.

Compared to the original RBV indexing scheme [8], we tradeoff speed to maximize the potential matches (candidate pool) and perform bitwise AND operation on the entire bit vector. This results in a larger number of false positives in the pool of candidate keypoint matches and consequently results in more computation. To reduce the cost of CPU computation (bitwise operation), we prune the number of processed dimensions during query evaluation to narrow the search space gradually while minimizing the number of false negatives. The number of dimensions to prune depends on the partition granularity (HCS) since these two parameters are coupled, that is, a change in one parameter will inevitably affect the other. We empirically evaluate the effects of varying HCS and the number of dimensions pruned on retrieval speed and accuracy in Section 7.

A summary of the process for querying the modified RBV index (henceforth referred as just RBV index) is as follows:

Require: Database of N items, Q local descriptors q of query, D dimensions i of $m_i - 1$ dividers and m_i bit-vectors, temporary resultant bit vectors R_Q (one for each q), temporary container $T[D]$.

```

for  $j = 1$  to  $Q$  do
    for  $k = 1$  to  $D$  do
         $T[k] = |q_{j\mu} - q_{jk}| / q_{j\mu}$ 
    end for
    Sort  $T$  in ascending order.
    for  $k = 1$  to  $T$  (or  $< T$  if pruned) do
        Get partition  $p$  using  $m_k - 1$  dividers; calculate  $R_j = R_j \& m_p$ .
    end for
    Perform  $L_2$  verification on matches (ON bits) in resultant bit vector.
end for
```

6 Evaluation Methodology

We demonstrate the effectiveness of our approach using a series of experiments. First, we evaluate the effectiveness of keypoint reduction on matching near-duplicate images, by varying the number of detected keypoints between 1, 400 (original number of keypoints), and using a subset of 500 and 100 most significant keypoints. We report the percentage of keypoint matches — relative to the keypoints in the query image — between a query image and each of its image alterations. For accurate evaluation, we use the sequential scan for the nearest-neighbor search on the collection of keypoints as both LSH and RBV indexes are approximate nearest-neighbor algorithms. For this experiment, due to exhaustive computation involved with using several keypoint thresholds, we use only 100 random queries on 10,000 images (Image Collection B as described below).

Second, we compare our approach — keypoint reduction with the modified RBV index — against the KSH system². We evaluate both efficiency and effectiveness of these approaches on a much larger collection of 20,000 images (Image

² The authors have provided their source code, allowing us to perform a direct comparison to their approach.

Collection A as described below). We use an identical framework as Ke et al. [11], the only differences being the index structure and the amount of PCA-SIFT features used. The original dataset³ is not used as it was not available, however, we generate a dataset using the same set of alterations as used in their work. A small number of images are selected at random from the collection as queries; the relevant answers are generated by transforming each query image using the set of alterations. Only altered versions of their respective original images are considered relevant answers; everything else in the collection is treated as noise. We evaluate our approach using the standard recall and precision metrics. All experiments are run on a two-processor Xeon 3 GHz machine with 3.8 GB of main memory running Linux 2.4.

Image Dataset. To generate our dataset, we select 250 images at random from Volume Twelve of the Corel Photo CD collection [2]; each image is altered using 40 alterations, creating a total of 10,000 images. We also include 10,000 images from the TRECVID 2005 collection, which consists of keyframes from various news broadcast. We scale all images to 512 pixels in the longer edge. Together with 10,000 altered images, we create a test collection of 20,000 images forming Image Collection A. Image Collection B is created using half of Image Collection A; consequently, we use 125 queries for this collection. As the PCA-SIFT algorithm does not use color information, all images are converted to greyscale after the altered image set is created. As in the work of Ke et al. [11] and Qamra et al. [14], the list of alterations are as follows : colorize (3), contrast (2), severe contrast (2), crop (3), frame (4), scale up (3), scale down (3), despeckle (1), saturation (4), intensity (6), shear (3), resize (3), and rotate (3). Note that the number in the parentheses indicate the alteration variants.

7 Results and Analysis

In Table II, the effects of keypoint reduction at every keypoint threshold value on 40 unique alterations are shown; the percentages are an average over 100 queries. Columns 1, 5, and 9 indicate the different types of alteration. The rest of the columns show the percentage of matching nearest-neighbors within the L_2 norm of 3,000 between the original image and its corresponding near-duplicate image. The average number of keypoints per image is close to 1,400. We experiment with a threshold of 500 and 100, reducing the average keypoints per image to 550 and 128, respectively. The last columns of each alteration (4, 9, and 12) show the percentage of matches using the same criterion with approximately 10% of the keypoints. The variation in percentages between image alterations in Table II is expected given that some alterations severely affect the properties of the local descriptors. The reason for the slight increase observed with threshold values of 500 and 100 — as compared to the original number of keypoints — is that the percentage of keypoint matches is relative to number of detected keypoints in the image (that is 550 and 128) indicating that most of the matching

³ <http://clweb02.pittsburgh.intel-research.net/yke/retrieval/>

Table 1. Percentage of keypoint matches within L_2 norm threshold at every level of reduction. Columns 1, 5, 9 indicate the different alterations (Alt). Keypoint thresholds of 500, and 100 are used. Default indicates the original number of keypoints (average of 1, 400).

Alt	Default	500	100												
1	92.4	94.1	95.0	11	76.5	82.0	85.3	21	7.3	5.2	5.9	31	86.7	87.5	89.5
2	88.8	89.8	91.2	12	57.1	54.7	57.3	22	80.0	79.7	79.1	32	83.1	88.0	89.2
3	90.5	92.0	92.5	13	49.7	48.3	49.7	23	80.3	79.5	78.3	33	74.5	82.9	81.4
4	76.9	80.3	80.4	14	48.4	46.4	45.4	24	79.7	79.0	78.2	34	39.8	42.8	43.4
5	77.6	76.3	76.5	15	54.0	52.7	55.8	25	89.6	91.5	92.3	35	34.3	27.9	28.2
6	58.6	63.3	64.1	16	56.9	60.9	62.0	26	90.9	93.0	93.8	36	24.2	13.1	9.0
7	53.7	58.3	58.8	17	55.7	58.5	57.3	27	92.5	94.2	95.1	37	73.4	71.9	74.9
8	47.6	52.3	53.2	18	57.6	60.1	61.8	28	92.1	94.0	95.1	38	55.4	60.1	50.7
9	42.9	47.8	49.1	19	47.1	37.9	42.1	29	90.7	92.5	93.3	39	48.1	50.1	48.4
10	71.5	71.1	75.8	20	19.4	15.8	18.0	30	83.4	84.0	87.0	40	43.4	35.7	29.4

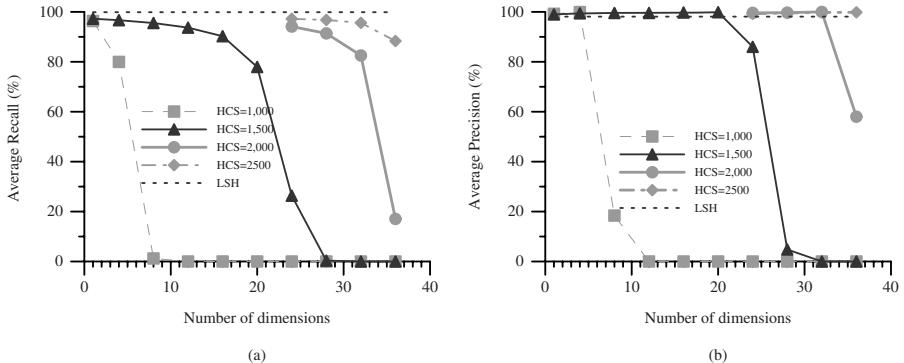


Fig. 1. Average (a) Recall and (b) Precision (over 250 queries) of the modified RBV index for variations of HCS and number of dimensions. LSH is the baseline.

keypoints within the L_2 -norm of the set of alterations share similar contrast values. This is an important finding as it is the criterion by which both the LSH and RBV indexes approximate matching keypoints. The relatively similar percentages of matching keypoints for different levels of reduction across all alterations, leads us to believe that even a small subset of keypoints is sufficient for this application. Subsequent experiments on the RBV index incorporate keypoint reduction with a threshold of 100 on the number of indexed features. In Figures 1(a) and 1(b), the effectiveness of the RBV index is measured using recall and precision, averaged over 250 queries. We experiment with varying the HCS parameter and the number of pruned dimensions; each increment of four dimensions is shown. LSH is used as a baseline with recall and precision of 99% and 98%, respectively. The highest observed recall and precision with RBV is 97% and 99% respectively, with an HCS of 1,500 when only one dimension is processed. We observe that even after processing 16 dimensions, recall remains at 91% and

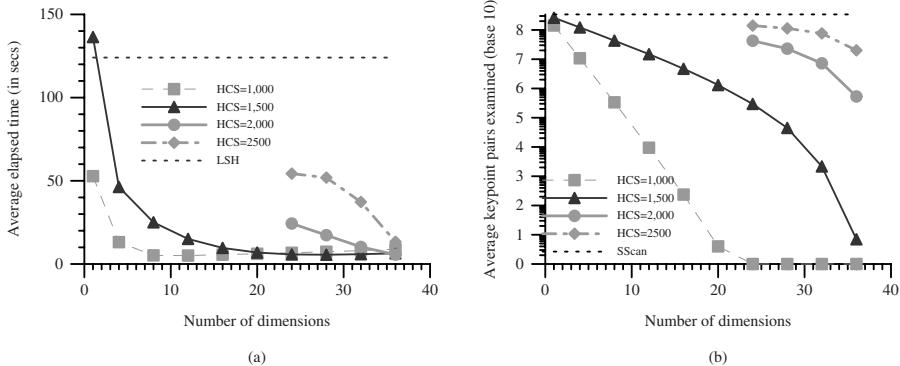


Fig. 2. (a) Average running time (over 250 queries) of the RBV index for variations of HCS and number of dimensions. LSH is the baseline. (b) Effectiveness (250 queries) of search space reduction of the RBV index. Sequential scan is the baseline.

precision at 98%. We do not experiment with smaller number of dimensions for HCS of 2,000, and 2,500 as we achieve near-perfect recall and precision after processing 24 dimensions. HCS of 2,500 achieves recall and precision of 88% and 99% respectively, even after processing all 36 dimensions. As expected, using HCS of 1,000, we observe a dramatic drop in recall and precision if more than 4 dimensions are processed, which implies that the boundaries in hypercube space is too “tight” resulting in high partition granularity. Hence, the choice of HCS is critical for the RBV index. We observe that given a large enough HCS, the drop in recall and precision is less abrupt, since the majority of the answers are still within the hypercube boundary of a single dimension resulting in fewer eliminated matches. We have thus shown that our modified RBV index is highly effective given a suitable HCS value.

Retrieval Efficiency. The timing results for query evaluation using the modified RBV index is presented in Figure 2a. The total running (elapsed) time for evaluating a single query is measured; all timings are averaged over the 250 queries. They are compared against the KSH baseline, which is observed to have an average running time of approximately 124 seconds. Since the KSH implementation of LSH, and our RBV implementation can be further optimized — in terms of in-memory data structures — we do not emphasize on the factors of improvement from the baseline. With our RBV index, the fastest recorded running time is approximately 9 seconds with an HCS of 1,500, and 16 dimensions; this was also observed to have high effectiveness. As expected, the running time reduces as more dimensions are processed; the pool of candidate matches becomes smaller, requiring fewer keypoints to be retrieved from disk. This is evident from the much higher running time of 136 seconds with HCS of 1,500 and processing only one dimension; this effectively reduces to an on-disk sequential scan. Using HCS of 1,000, we observe that there is a slight increase in running time from 5 to 8 seconds when the number of dimensions is more than 12. We

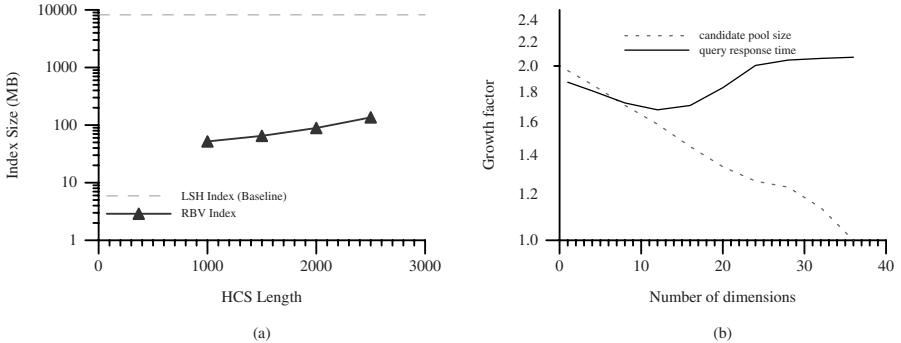


Fig. 3. (a) Effects of HCS on the RBV index size. LSH is the baseline. (b) Growth factors of candidate pool size and query run-time between image collections B and A (observed using HCS of 1,500).

believe this is due to the increased cost of processing (CPU operations required for bitwise ANDing and fetching bit vectors from disk) more dimensions without a corresponding decrease in the number of keypoint pairs. Finally, for HCS of 3,000 the running time for processing all 36 dimensions is comparable to that of the HCS of 1,500 while still showing high effectiveness.

Further Studies. It is instructive to examine the effectiveness of the RBV index in reducing the search space. Figure 2b shows the total number of keypoints being processed by the RBV index. This is an independent study on the RBV index on the same collection; no comparisons are made against the LSH index since we do not experiment with keypoint reduction on the KSH system. Instead, we compare it to sequential scan to illustrate the reductions in search space using the RBV index. All numbers are reported as an average over 250 queries.

The sequential scan always requires the worst-case number of keypoints, as it performs a brute-force search to find k -nearest-neighbors within the L_2 threshold. Using HCS of 1,500, the candidate pool is quickly reduced from approximately 260 million to 120 million keypoints after processing only 4 dimensions. For 16 dimensions only 4 million keypoints remain in the candidate pool. Naturally, a smaller number of candidate matches translates to higher efficiency, since fewer keypoints need to be fetched and examined. Indeed, this result shows that the RBV index is effective at reducing the search space using only a few dimensions, while minimizing the number of false negatives. As in Figure 3a, we show the effects on index size using different HCS values; these sizes are observed for an index of 20,000 images. This clearly shows that HCS dictates the number of partitions, which determines the number of bit vectors that are stored on disk. It is also interesting to note that using the baseline method without keypoint reduction as described in the work of Ke et al. [11], the size of the index is considerably larger than that of RBV. Finally, as shown in Figure 3b, we observe that the candidate pool size increase by only a factor of 1.4 (using 16 dimensions),

even though the collection size increases by a factor of 2 (Image Collections B to A); the slight growth of the query response time is attributed to the increase in in-memory bitwise processing.

8 Conclusion

We have presented an approach to near-duplicate image detection with pruned SIFT keypoints (using PCA-SIFT local descriptors) using our proposed modified RBV indexing scheme. An almost lossless retrieval performance is observed using only 10% of the original keypoint features, thereby reducing index size and improving scalability.

We show that, unlike the original approach that was initially designed for negative queries, near-perfect effectiveness can be achieved using our modified approach for positive queries as well. We demonstrate that this indexing scheme performs as well as the KSH system in terms of effectiveness and runs in a little under ten seconds on average for a single query, on a collection of 20,000 images. Importantly, the RBV index is — highly compact — over 100 times smaller than that of the original LSH index as used in the KSH system.

As observed in our experiments, our approach has shown the highest efficiency — a factor-of-12 speed-up over the KSH system — thus far, in this domain. Hence, this approach offers a promising and viable alternative indexing strategy to the predominant LSH approach. We intend to explore the limitations and scalability of these schemes in future work.

Acknowledgments

This project was supported by Australian Research Council. We thank Justin Zobel for his suggestions.

References

1. C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
2. Corel Corporation. Corel professional photos CD-ROMs, 1994.
3. M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.
4. J. J. Foo and R. Sinha. Pruning sift for scalable near-duplicate image matching. In *Proc. ADC Australian Database Conference*, January 2007.
5. J. J. Foo, R. Sinha, and J. Zobel. Discovery of image versions in large collections. In *Proc. MMM Int. Conf. on Multimedia Modelling*. Springer, Januuary 2007.
6. A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. VLDB Int. Conf. on Very Large Data Bases*, pages 518–529, Edinburgh, Scotland, UK, September 1999. Morgan Kaufmann.

7. J. Goldstein, J. C. Platt, and C. J. C. Burges. Indexing high dimensional rectangles for fast multimedia identification. Technical report, Microsoft Research, Redmond, WA, USA, 2003.
8. J. Goldstein, J. C. Platt, and C. J. C. Burges. Redundant bit vectors for quickly searching high-dimensional regions. In *Deterministic and Statistical Methods in Machine Learning, First International Workshop, Sheffield, UK, September 7-10, 2004, Revised Lectures*, pages 137–158. Springer, 2004.
9. K. Grauman and T. Darrell. Efficient image matching with distributions of local invariant features. In *Proc. CVPR Int. Conf. on Computer Vision and Pattern Recognition*, pages 627–634, June 2005.
10. Y. Ke and R. Sukthankar. PCA-sift: A more distinctive representation for local image descriptors. In *Proc. CVPR Int. Conf. on Computer Vision and Pattern Recognition*, pages 506–513, Washington, DC, USA, June–July 2004. IEEE Computer Society.
11. Y. Ke, R. Sukthankar, and L. Huston. An efficient parts-based near-duplicate and sub-image retrieval system. In *Proc. MM Int. Conf. on Multimedia*, pages 869–876, New York, NY, USA, October 2004. ACM Press.
12. D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2):91–110, 2004.
13. K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. In *Proc. CVPR Int. Conf. on Computer Vision and Pattern Recognition*, pages 257–263, June 2003.
14. A. Qamra, Y. Meng, and E. Y. Chang. Enhanced perceptual distance functions and indexing for image replica recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(3):379–391, 2005.

OLYBIA: Ontology-Based Automatic Image Annotation System Using Semantic Inference Rules*

Kyung-Wook Park, Jin-Woo Jeong, and Dong-Ho Lee^{**}

Department of Computer Science and Engineering, Hanyang University
Ansan-si, Gyeongki-do 426-791, South Korea
`{kwpark, jwjeong, dhlee72}@cse.hanyang.ac.kr`

Abstract. One of the big issues facing current content-based image retrieval is how to automatically extract the high-level concepts from images. In this paper, we present an efficient system that automatically extracts the high-level concepts from images by using ontologies and semantic inference rules. In our method, MPEG-7 visual descriptors are used to extract the visual features of image, and the visual features are mapped to semi-concepts via the mapping algorithm. We also build the visual and animal ontologies to bridge the semantic gap. The visual ontology allows the definition of relationships among the classes describing the visual features and has the values of semi-concepts as the property values. The animal ontology can be exploited to identify the high-level concept in an image. Also, the semantic inference rules are applied to the ontologies to extract the high-level concept. Finally, we evaluate the proposed system using the image data set including various animal objects and discuss the limitations of our system.

Keywords: high-level concepts, ontologies, semantic inference rules, MPEG-7 visual descriptors, semantic gap.

1 Introduction

In the past decade, due to the rapid growth of the Internet and mobile device, the amount of available multimedia contents has explosively increased both in numbers and in size. A number of users and applications are available over the Internet for browsing and searching the collections of multimedia contents.

Most users often tend to use the abstract notion involved in an image when searching for an image. However, traditional image retrieval approaches have a lot of problems that make it difficult to search for images using high-level concepts. Therefore, there are needs for the development of tools that support the semantic retrieval for a large image database.

Current image retrieval techniques can be classified into two main categories: keyword-based and content-based image retrieval. In the former approach, image

* This work was supported by Korea Research Foundation Grant funded by the Korea Government(MOEHRD) (KRF-2006-521-D00457).

** Corresponding author.

contents have to be manually annotated by domain experts using a restricted vocabulary [1,2]. Although this approach is able to provide the functionality of the semantic retrieval, this suffers from several limitations in a large multimedia database. Since images have to be manually annotated by annotators, this method needs a large amount of manual effort required in generating annotations for large image collections. Consequently, it is likely that an image is annotated with only one or a small subset of possible semantic interpretations.

Content-based image retrieval (CBIR) systems retrieve images based on the visual similarity of the visual features (i.e., color, texture, and shape etc.) that are semi-automatically extracted and stored [3,4]. Although this method is less time-consuming and provides more user-friendly access on images than keyword-based approach, the problem is that the visual similarity does not necessarily mean the semantic similarity. For example, if a user requests an image with a ‘red rose’ to a CBIR system, it is likely to answer images with a ‘red ball’ because it returns the result based on the visual similarity. That is, there may be a gap between the visual features and the high-level concepts of an image. This problem is called ‘*Semantic Gap*’ by researchers of this field.

To overcome such drawbacks, a few researches have been done on using ontologies for the retrieval of visual resources such as image and video [5-7]. Ontology is a type of background knowledge that defines all of the important categories of concepts that exist in a specific domain, and the relationships between them. In early researches on the image retrieval using ontology, ontologies are just used for assisting manual annotation [5,6]. These ontologies are unsuitable for automatic annotation since they contain little visual information about the concepts they describe. Therefore, a few researches have been investigating how to automatically annotate the high-level concepts in the image by using ontologies [7]. They employed ontologies to annotate the high-level concepts automatically from the visual features which are extracted by various image processing techniques. However, since they employed too simple ontologies, their works still have several problems.

In this paper, we propose a new image annotation system (so-called OLYBIA: OntoLogY-Based Image Annotation system) where the visual features are mapped into the semi-concepts and the high-level concepts are automatically extracted by applying the inference rules to the visual and animal ontologies. In our work, the high-level concept implies the object name in an image (e.g., tiger, eagle, etc.) and the values of semi-concepts that are simple texts assigned according to the quantity of the visual features, such as ‘high’ or ‘low’, are used to abstract the values of the visual features. The visual ontology focuses on describing the visual features of an image. In particular, since it is based on the characteristic that the visual features are not restricted to a specific domain, it is possible to apply for various domains such as medical images, artwork images, and etc. We also construct the animal ontology representing animal taxonomy. The reason that we concentrate on animal taxonomy in building the animal ontology is that most users are interested in the kind of animal when searching for animal images. Finally, in order to identify the high-level concepts, the semantic inference rules are applied to the visual and animal ontologies.

The rest of this paper is organized as follows: In Section 2, we briefly review related work. Section 3 describes the architecture of OLYBIA and the methods that extract the high-level concept from an image. Section 4 introduces the design and implementation of OLYBIA, and Section 5 shows the experiment results. Finally, we conclude our work and discuss our future plans in Section 6.

2 Related Work

In recent years, a few researches have been done on using ontologies for the retrieval of visual resources such as image and video [5-7].

As an early research on image annotation and retrieval using ontologies, A. T. Schreiber *et al.* [5] have been studied the use of background knowledge contained in ontologies to index and search the collections of images. In particular, to describe the semantic information called *subject matter description*, they constructed a domain-specific ontology for the animal domain that provides the vocabulary and background knowledge describing features of the photo's subject matter. However, although their work shown the benefits of image annotation obtained by using ontologies, they did not reduce the burden of annotators because ontologies are just used for assisting manual annotation.

In [6], the authors proposed a hierarchical video content description and summarization strategy supported by a novel joint semantic and visual similarity. In order to describe the video content accurately, they used the low-level visual features that are extracted by various video processing techniques, and the semantic features that are manually annotated using a *video content description ontology*. Based on the similarity of these visual and semantic features, they constructed a hierarchical video content structure by merging and grouping a small video unit into a bigger video unit. Although they proposed a method using an ontology to describe the video content semantically, this work is also performed by manual annotation.

Vasileios Mezaris *et al.* [7] proposed the method which uses an *object ontology* and *intermediate-level descriptor values* to describe semantic information. First, an image is segmented to a number of regions by using an image segmentation algorithm, and then the visual features of each region are automatically extracted. The extracted visual features are mapped to human-readable intermediate-descriptor values. Finally, the object of an image is identified by using the object ontology which has the name of each object as its top-level class and the intermediate-level descriptor values as its property values. Here, the object ontology is a specification for a specific object. For example, ‘tiger’ object is defined as *Luminance* = {high, medium}, *green-red* = {red low, red medium}, *blue-yellow* = {yellow medium, yellow high} and *size* = {small, medium}, where the intermediate-level descriptor values such as ‘high’ and ‘red low’ are defined based on the visual features. However, this method has several limitations as follows: First, the object ontology only takes into account the subsumption relationship between the object and the visual feature classes. However, to describe the image content efficiently and accurately, it is necessary to define the relationship among the objects, and the relationship among the visual feature classes as well as the relationship between the object and its visual features. Second, since the object ontology has to be built for each object individually, it must be reorganized every time a new object comes from domain experts. That is,

since their approach did not employ any inference rule for extracting the high-level concept, the object ontology must be reorganized as a new object is added. In general, semantic inference rules can be used to derive the new knowledge from existing knowledge in a domain.

In order to address these problems, we will infer the high-level concepts by applying the semantic inference rules to the visual and animal ontologies. In particular, since such rules can be shared and collaboratively modified as the domain understanding changes, it is not necessary that the domain ontology is rebuilt when a new object comes from domain experts or existing knowledge is modified. That is, we have only to redefine the semantic inference rules.

3 The Architecture of OLYBIA

In this section, we first introduce the architecture of OLYBIA and explain the methods for mapping the visual features to the semi-concepts in detail. And then, we describe the method for detecting the semantic inference rules automatically via the training data set including animal objects of the same kind.

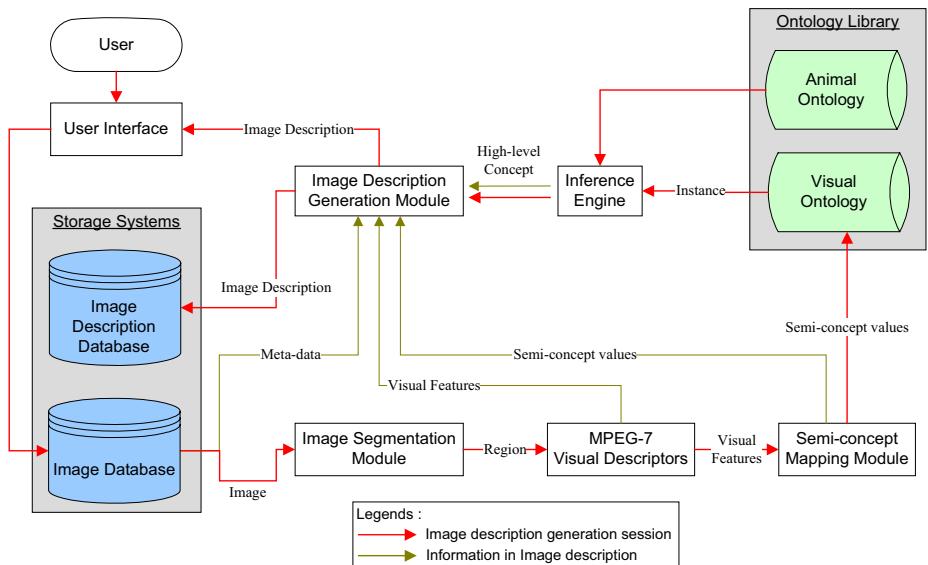


Fig. 1. The architecture of OLYBIA

The overall architecture of OLYBIA is shown in Figure 1. The operation of OLYBIA is executed in the order as follows: First, in image segmentation module, the region of interest (ROI) is segmented from an image using the algorithm proposed in [8]. Then, the MPEG-7 visual descriptors extract the visual features from ROI. For this, we use the edge histogram descriptor (EHD) for the texture feature, the contour-based shape descriptor (Contour-SD) for the shape feature, and the color structure descriptor (CSD) for the color feature. The extracted visual features are mapped to the

semi-concepts via the semi-concept mapping module that will be explained in Section 3.1. These semi-concepts are stored as the property values of an instance of the visual ontology. Finally, the inference engine extracts the high-level concepts by applying the semantic inference rules to the visual and animal ontologies. By using these high-level concepts, the image description generation module generates the final image description together with the additional information, and then stored them into an image description database for the efficient image management and retrieval.

3.1 Semi-concept Value Mapping

MPEG-7 is a standard for describing multimedia content published by the Moving Picture Experts Group (MPEG) [9] and broadly offers a number of tools which describe the multimedia contents in various aspects. According to Spyrou E. *et al.* [10], it is better to exploit several visual features than only one visual feature. Therefore, we use the EHD, CSD, and Contour-SD to extract representative visual features that are mapped to the semi-concept. As mentioned in Section 1, the values of semi-concepts are simple keywords which are automatically assigned by matching the quantity of visual features into keywords according to a specific range.

For the color feature, we use the CSD that represents an image by both the color distribution of an image (similar to a *color histogram*) and the local spatial structure of the color [11]. The color histogram is used for calculating the dominant colors DC_{0-1} and the colorfulness $diff_{0-1}$ for DC_{0-1} , and then these are mapped into the semi-concepts S_DC_{0-1} and S_Diff_{0-1} , respectively as follows:

Algorithm 1. *Semi-concept value mapping algorithm*

INPUT: an input image I .

OUTPUT: a set of semi-concept values that represent the color feature of I .

1. A 256-bin color histogram is extracted from the input image I , and then bins are unified to a 128-bin color histogram for more efficient computation.
2. The 128-bin color histogram h is defined via five subspaces, i.e., S_m , $m=0,\dots,4$, in the HMMD color space. S_{0-2} represent the color by dividing *hue* into 8 uniform intervals and *sum* into 4 uniform intervals, giving 8x4 cells in subspaces, respectively. On the other hand, we do not consider S_3 because it represents the color information differently from S_{0-2} . S_4 represents the gray scale by dividing *hue* into 1 uniform intervals and *sum* into 16 uniform intervals. The semi-concepts S_DC_0 and S_DC_1 for DC_0 and DC_1 are defined by

$$C_{j \in \{0, \dots, 7\}} = \sum_{i=0}^2 S_{ij},$$

$$DC_0 = \text{Max}(tot), DC_1 = \text{Max}(tot - DC_0), \text{ where } tot = \{C_0, \dots, C_7\}$$

$$S_DC_{k \in \{0,1\}} = \begin{cases} \text{Red-Orange} & \text{if } DC_k = C_0 \\ \dots & \dots \\ \text{Pink} & \text{if } DC_k = C_6 \\ \text{Red} & \text{if } DC_k = C_7 \end{cases}$$

where sub-colors C_j denote the sum of the values of all the bins belonging to the j th cell S_{ij} in the subspaces S_i .

3. The semi-concepts S_diff_k for $diff_o$, $diff_i$ are calculated as follows:

$$\text{for each } DC_k = C_j, \quad diff_k = \text{Max}(C_j^{S_{ij}}), \text{ where } C_j^{S_{ij}} = \bigcup_{i=0}^2 S_{ij}$$

$$S_diff_k = \begin{cases} \text{high,} & \text{if } diff_k = S_{0j} \\ \text{medium,} & \text{if } diff_k = S_{1j} \\ \text{low,} & \text{if } diff_k = S_{2j} \end{cases}$$

4. The semi-concept S_DG for gray scale can be also calculated in a similar manner.

The semi-concept values for the other visual features, EHD and Contour-SD, can be also extracted automatically using the algorithms similar to Algorithm 1.

EHD represents the local edge distribution in an image. Thus, it is useful for retrieving natural images with non-uniform textures [11,12]. For the images containing homogeneous object, we found the fact that two edge types having the maximum bin size in the global-edge histogram are almost similar. Therefore, we only consider the global-edge histogram g_EH . The semi-concepts $S_EH_{0\sim 1}$ for two representative edges are as follows:

$$g_EH = \{E_0, E_1, E_2, E_3, E_4\}, \text{ where edge types } E_{k \in \{0,1,2,3,4\}} = \sum_{i=0}^3 \sum_{j=0}^3 e_k^{h_{ij}}$$

$$E'_0 = \text{Max}(g_EH), \quad E'_1 = \text{Max}(g_EH - E'_0)$$

$$S_EH_{l \in \{0,1\}} = \begin{cases} \text{Vertical,} & \text{if } E'_k == E_0 \\ \text{Horizontal,} & \text{if } E'_k == E_1 \\ \text{45diagonal,} & \text{if } E'_k == E_2 \\ \text{135diagonal,} & \text{if } E'_k == E_3 \\ \text{Nondirectional,} & \text{if } E'_k == E_4 \end{cases}$$

where the local-edge histogram, $h_{ij} = \{e_0^{h_{ij}}, e_1^{h_{ij}}, e_2^{h_{ij}}, e_3^{h_{ij}}, e_4^{h_{ij}}\}$, is a local edge distribution corresponding to row i x column j subimage in an image I .

The semi-concept $S_Contour_{0\sim 3}$ for Contour-SD can be also calculated in a similar manner to the above. Due to the lack of space, we omit explaining the mapping procedure of Contour-SD to its corresponding semi-concepts $S_Contour_{0\sim 3}$.

As a result, these semi-concepts extracted by Algorithm 1 are stored as the property values of an instance of the visual ontology that will be explained in next section.

3.2 Visual and Animal Ontologies

The W3C has established the OWL web ontology language on the basis of RDF. We make use of OWL that offers the vocabulary for describing properties and classes in order to describe the visual and animal ontologies.

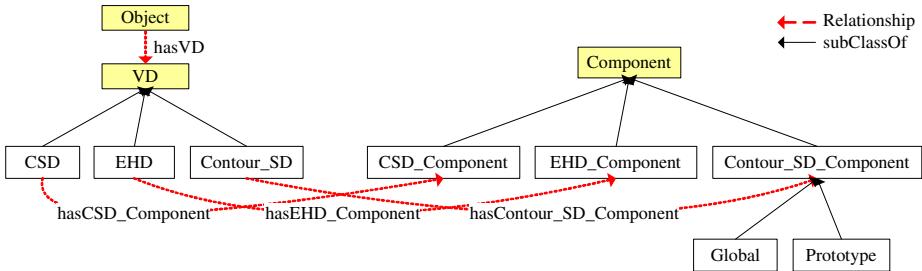


Fig. 2. Class hierarchy and relationships of the classes in the visual ontology

Table 1. The definition of the classes in the visual ontology

Class	Property	Definition
Object	hasVD VD	Describing the object in an image
Component	None	The component of MPEG-7 visual descriptors
VD	None	Describing the MPEG-7 visual descriptors
CSD	\exists hasCSD_Component CSD_Component	Describing the CSD of the MPEG-7 visual descriptors
EHD_Component	S_EH0_Value, S_EH1_Value	The semi-concept values of the EHD

Figure 2 shows the part of the visual ontology. As shown in Figure 2, it is made up of various classes and relationships among them. Table 1 shows the definition of some classes in the visual ontology. For example, as depicted in Table 1, *Object* class, the top-level class, describes the object in an image and is concerned with visual descriptor (*VD*) class by *hasVD* relationship. In the case of the classes that are not defined in Table 1 (e.g., the *CSD_Component* class), they are also similar to the definition of the sibling classes.

```
1 <Object rdf:ID = "Object_1">
2   <hasVD rdf:resource = "#EHD_1"/>
3   <hasVD rdf:resource = "#CSD_1"/>
4   <hasVD rdf:resource = "#Contour_SD_1"/>
5 </Object>
6 <EHD rdf:ID = "EHD_1">
7   <hasEHD_Component>
8     <EHD_Component rdf:ID = "EHD_Component_1">
9       <S_EH0_Value rdf:datatype = http://www.w3.org/...#String> Horizontal <...>
10      <S_EH1_Value rdf:datatype = http://www.w3.org/...#String> 45diagonal <...>
11    </EHD_Component>
12  </hasEHD_Component>
13 </EHD>
14 <CSD rdf:ID = "CSD_1">
15   <hasCSD_Component>
16     <CSD_Component rdf:ID = "CSD_Component_1">
```

```

17 <S_DC0_Value rdf:datatype =http://www.w3.org/...#String>Yellow-Green</..>
18 <S_DC1_Value rdf:datatype =http://www.w3.org/...#String>Sky-Blue</..>
19 ...

```

The above OWL document denotes the part of the visual ontology for the instance *Object_1*. In line 1~5, *Object_1* is concerned with the instances *EHD_1*, *CSD_1*, and *Contour_SD_1* by *hasVD* relationship. In line 6~13, *EHD_1* has the string values ‘Horizontal’, ‘45diagonal’ for the semi-concepts *S_EH₀* and *S_EH₁*, respectively. Also, in line 14~19, *CSD_1* has the string values ‘Yellow-Green’, ‘Sky-Blue’ for the semi-concepts *S_DC₀* and *S_DC₁*, respectively.

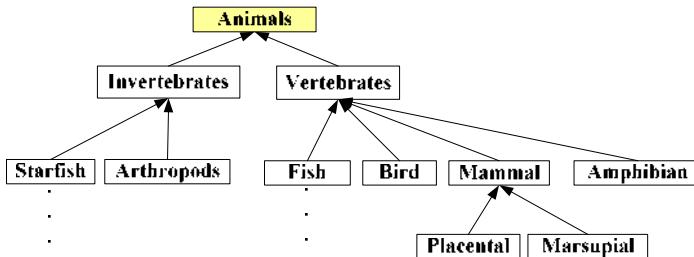


Fig. 3. The class hierarchy of the animal ontology

We also constructed the animal ontology representing animal taxonomy since the evaluation domain is animal images. Figure 3 depicts the class hierarchy of the animal ontology that is used to infer the high-level concepts in animal images. Note that the goal of animal ontology is only to provide the type of an object. Therefore, the relationships among the classes are not defined. In fact, we do not describe all kinds of animal terms because it is very difficult to construct the animal ontology consisting of all kinds of animal terms.

3.3 Semantic Inference Rules

The aim of inference is to derive the new knowledge by applying the inference rules to existing knowledge in a specific domain. Currently, various rule engines for OWL reasoning have been proposed [13,14]. We use ‘Bossam’ rule engine [15] which provides a rule language called ‘Buchingae’ for OWL reasoning.

The key idea for automatic detection of the semantic inference rules is based on the observation that the visual features of the same objects are very similar to each other. Based on this characteristic, the most common values of the semi-concepts for an object are defined as a rule’s terms.

In our work, we used eleven categories of distinct semi-concepts for each visual feature. That is, there are five categories for the color feature, {*S_DC₀*, *S_DC₁*, *S_diff₀*, *S_diff₁*, *S_DG*}, two categories for the texture feature, {*S_EH₀*, *S_EH₁*}, and four categories for the shape features, {*S_Contour₀*, *S_Contour₁*, *S_Contour₂*, *S_Contour₃*}. Consequently, the algorithm for detecting the semantic inference rules is to find the most frequently appeared terms for each semi-concept category on the

training image data set, $I = \{I_0, \dots, I_n\}$, with the same object. The algorithm for detecting the inference rule is as follows:

Algorithm 2. Automatic detection of the semantic inference rules

INPUT: a set of the semi-concept values from all images in the training data set I .

OUTPUT: a set of terms that will be used in the inference rules.

1. We extract the semi-concepts from all images in I and count the number of images with the same semi-concept values. That is, we calculate the distribution of semi-concept values, $SH = \{sh_0, \dots, sh_{10}\}$, for each semi-concept category. As mentioned above, there are eleven categories for distinct semi-concepts. Thus, the total number of the elements of SH is 11.
2. The standard deviations $\sigma_{x \in \{0, \dots, 10\}}$ for each $sh_{x \in \{0, \dots, 10\}}$ are calculated. In case of that the value of $\sigma_x < \tau$, the procedure is terminated because this means that the images have too various semi-concept values to discover the most frequently appeared terms for a specific rule.
3. In case of that the value of $\sigma_x \geq \tau$, the most common values among the values of sh_x become the element of the rule set CR_x .

In Algorithm 2, if the standard deviations σ_x for sh_x is less than a threshold (τ), it means that the training images with the same object have too various semi-concept values to discover the most common semi-concept values that are used in the inference rules. Actually, though the experiments, we found that OLYBIA shows a good performance when τ is about 0.4.

By Algorithm 2, we can decide the terms that are used in the semantic inference rule. For example, let us assume that the most common semi-concept values for each semi-concept category on the images containing the ‘tiger’ object are as follows:

$sh_0 (=S_DC_0) = \text{'Red-Orange'}$, $sh_2 (=S_diff_0) = \text{'Medium'}$, ... , $sh_4 (=S_EH_0) = \text{'Nondirectional'}$, ...

Then, the rule for identifying the ‘tiger’ object is as follows.

```
If vdo : object(?x) ∧
  vdo : hasVD(?x, ?y) ∧ vdo : hasEHD_Component(?y, ?z) ∧
  vdo : S_EH_0_Value(?z, "Nondirectional") ∧ vdo : S_EH_1_Value(?z, "45diagonal") ∧
  ...
  vdo : hasCSD(?x, ?a) ∧ vdo : hasCSD_Component(?a, ?b) ∧
  vdo : S_DC_0_Value(?b, "Red-Orange") ∧ vdo : S_diff_0_Value(?b, "Medium")
⇒ ani : Tiger(?x)
```

We used a training data set consisting of about 1,300 images to find out the inference rules identifying 12 kinds of animal objects (i.e., tiger, cheetah, eagle, penguin, etc.). However, we could not discover the rules for the ‘penguin’ and

‘avocet’ objects. It is possible to extract different visual features for the same object because the visual features are sensitive to the physical environment (e.g., camera angle, light, etc.).

4 Design and Implementation

We have designed the user interface to allow the users to browse all of the image information more intuitively. Although we mainly focused on the automatic extraction of high-level concepts from an image, we have built the system generating the image description which includes visual features and additional information as well as high-level concepts to satisfy various requirements of the users on the image management and retrieval.

Figure 4 depicts the snapshot of OLYBIA interface. The interface of OLYBIA can be functionally divided into 4 main parts: 1) Meta-data description panel represents the metadata information such as a creation time, user comment, and etc. 2) Visual information panel describes the visual features (i.e., CSD, EHD, and Contour-SD) of the corresponding image that are extracted by the MPEG-7 visual descriptors. 3) Physical information panel represents the physical information of the corresponding image, such as the type and size of an image. 4) Semantic information panel shows the semi-concept values and the high-level concept that are automatically extracted by Algorithm 1 and 2. The image description including all information of the corresponding image is generated as an OWL document and stored into an image description database.

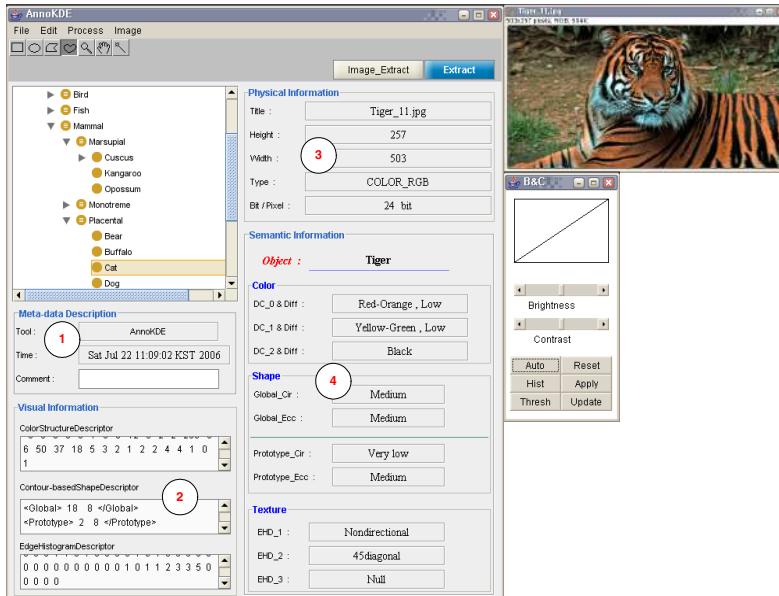


Fig. 4. The snapshot of OLYBIA interface

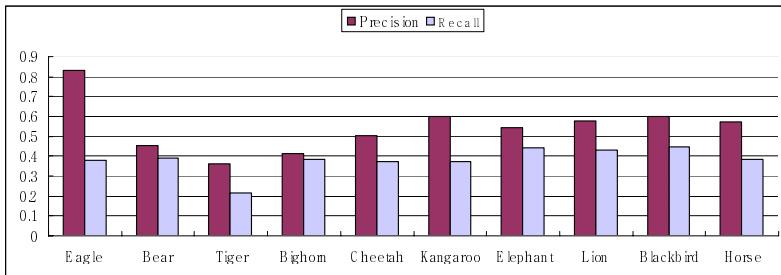


Fig. 5. The accuracy of image annotations by the semantic inference rules

5 Experimental Evaluation

Our system is evaluated on Corel image database which consists of about 2,500 color images with 12 semantic categories. These categories are *Polar bear*, *Cheetah*, *Eagle*, *Elephant*, *Bighorn*, *Lion*, and etc. For the performance evaluation, we measured the accuracy of the image annotation using precision and recall parameters which are calculated by equation (1).

$$p = \text{Num}_{\text{AnnoRel}}/\text{Num}_{\text{Anno}}, \quad r = \text{Num}_{\text{AnnoRel}}/\text{Num}_{\text{Rel}} \quad (1)$$

where $\text{Num}_{\text{AnnoRel}}$ means the number of relevant images annotated, Num_{Anno} is the total number of images annotated, and Num_{Rel} is the number of relevant images in the image data set.

Figure 5 shows the results of our experiments. Through the experiments, we can deduce a few facts. First, we could know that average precision is higher than average recall. Since the goal of the image annotation is how to more accurately describe the image content in an image database, this result is desirable in aspect of the image annotation. Second, we could discover that, if an object has distinct visual features among objects, its corresponding images are likely to be annotated more accurately. As depicted in Figure 5, the precision of ‘Eagle’ object is relatively high as compared with those of the other objects because it has very different visual features from the other objects. On the other hand, the error may arise for objects, such as ‘Tiger’ and ‘Cheetah’ objects, that have similar visual features.

6 Conclusion

In this paper, we proposed a new image annotation system called OLYBIA where the visual features are mapped into semi-concepts and the high-level concepts are automatically extracted by applying the inference rules to the visual and animal ontologies. In particular, since the visual ontology is based on the characteristic that the visual features are not restricted to a specific domain, it is possible to apply for various domains such as medical image, artwork image, and so on.

As compared with other image annotation systems, OLYBIA has advantages as follows: First, since the entire progress is automatically accomplished, we can quickly

describe a variety of information including high-level concept. Second, although OLYBIA is designed for the animal domain, it is possible to apply various domains without serious modification because the visual ontology and the semantic inference rules can be reused.

However, OLYBIA has also several limitations. First, an image may contain more than two objects. Therefore, we need to describe more than one object and the relationship among them. For example, in the case of the semantic query such as '*Crocodile bird on the crocodile*', we have to define the spatial relationship between the '*crocodile*' and '*crocodile bird*' objects. We are now studying how to define the relationships among various objects to extract rich semantic information from an image. Second, in order to obtain sophisticated inference rules, the algorithm for detecting the inference rules have to be more elaborated upon. Our future work will focus on addressing these problems.

References

1. W. E. Mackay: EVA: An experimental video annotator for symbolic analysis of video data. SIGCHI Bulletin, Vol. 21 (1989) 68-71
2. Eitetsu Oomoto and Katsumi Tanaka: OVID: Design and Implementation of a Video-Object Database System. IEEE Trans. On Knowledge and Data Engineering, Vol. 5 (1993) 629-643
3. John R. Smith and Shih-Fu Chang: VisualSEEK: a fully automated content-based image query system. ACM Multimedia 96 (1996)
4. Chad Carson, Megan Thomas, Serge Belongie, Joseph M. Hellerstein and Jitendra Malik: Blobworld: A System for Region-Based Image Indexing and Retrieval. Third International Conference on Visual Information Systems (1999)
5. A. T. Schreiber, B. Dubbeldam, J. Wielemaker, and B. J. Wielinga, "Ontology-based photo annotation", IEEE Intelligent Systems (2001) 66-74.
6. Xingquan Zhu, Jianping Fan, Ahmed K. Elmagarmid, Xindong Wu, "Hierarchical video content description and summarization using unified semantic and visual similarity", Multimedia Syst. 9(1) (2003) 31-53
7. Vasileios Mezaris, Ioannis Kompatsiaris, and Michael G. Strintz, "Region-based Image Retrieval using an Object Ontology and Relevance Feedback", EURASIP JASP, 2004
8. M. Jacob, T. Blu and M. Unser: Efficient energies and algorithms for parametric snakes. IEEE Transactions on Image Processing, Vol. 13 (2004) 1231-1244
9. ISO/IEC 15938-5 FDIS Information Technology: MPEG-7 Multimedia Content Description Interface - Part 5: Multimedia Descriptin Schemes. (2001)
10. Spyrou E., Le Borgne H., mailis T., Cooke E., Avrithis Y. and O'Connor N: Fusing MPEG-7 visual descriptors for image classification. ICANN 2005 (2005) 11-15
11. BS Manjunath, Philippe Salembier and Thomas Sikora: Introduction to MPEG-7. (2002)
12. D.K. Park, Y.S. Jeon, C.S. Won and S. -J. Park: Efficient use of local edge histogram descriptor. ACM International Workshop on Standards, Interoperability and Practices, Marina del Rey, California, USA (2000) 52-54
13. Hewlett-Packard: Jena Semantic Web Framework. <http://jena.sourceforge.net/> (2003)
14. UMBC: F-OWL: An OWL Inference Engine in Flora-2. <http://fowl.sourceforge.net>
15. Minsu Jang and Joo-Chan Sohn: Bossam: An Extended Rule Engine for OWL Inferencing. RuleML 2004 (2003) 128-138.

OntoDB: An Ontology-Based Database for Data Intensive Applications

Hondjack Dehainsala, Guy Pierra, and Ladjel Bellatreche

LISI/ENSMA, Téléport 2, 1, ave. Clément Ader 86960 Futuroscope - France
`{dehainsala,pierra,bellatreche}@ensma.fr`

Abstract. Recently, several approaches and systems were proposed to store in the same database data and the ontologies describing their meanings. We call these databases, ontology-based databases (OBDBs). Ontology-based data denotes those data that represent ontology individuals (i.e., instance of ontology classes). To speed up query execution on the top of these OBDBs, efficient representations of ontology-based data become a new challenge. Two main representation schemes have been proposed for ontology-based data: vertical and binary representations with a variant called hybrid. In these schemes, each instance is split into a number of tuples. In this paper, we propose a new representation of ontology-based data, called *table per class*. It consists in associating a table to each ontology class, where all property values of a class instance are represented in a same row. Columns of this table represent those properties of the ontology class that are associated with a value for at least one instance of this class. We present the architecture of our ontology-based databases and a comparison of the effectiveness of our representation scheme with the existing ones used in Semantic Web applications. Our benchmark involves three categories of queries: (1) targeted class queries, where users know the classes they are querying, (2) no targeted class queries, where users do not know the class(es) they are querying, and (3) update queries.

1 Introduction

Nowadays, ontologies are largely used in several research and application domains, such as, Semantic Web, information integration, e-commerce, etc. Actually, several tools for managing (building, inferring, querying, etc.) ontology data and ontology-based data (also called ontology individuals or ontology class instances) are available (e.g., Protégé 2000). Usually, ontology-based data manipulated by these tools are stored in the main memory. Thus, for applications manipulating a large amount of ontology-based data, query performance becomes a new issue. Over the last five years, several approaches have been proposed for storing both ontologies and ontology-based data in a database in order to get benefit of the functionalities offered by DBMSs (query performance, data storage, transaction management, etc.) [2][3][4][8][10][11] (see [15] for an extensive comparison). We call this kind of databases, ontology-based databases (OBDBs).

Two main OBDB structures for storing ontology and ontology-based data were proposed: *single table approach* and *dual schemes approach*. In the single table approach [13, 8, 10], the description of classes, properties and their instances are stored in a single table called *vertical table* [1]. The schema of this table has three columns: *subject*, *predicate*, *object*, representing instance identifier, property of an instance and value of an instance, respectively. This approach is simple to implement and its structure may be used both for the ontology and for instance data. Therefore, tools (inference engine, APIs, etc.) developed for storing ontologies can also be used for processing instances data. To ensure a high performance of queries, each column shall be indexed and the predicate column shall be clustered [1]. Materialized views can also be used [11]. Its main drawbacks are: (1) an extra storage cost (for storing indexes), (2) a maintenance overhead, and (3) its inefficiency for processing large join queries [2].

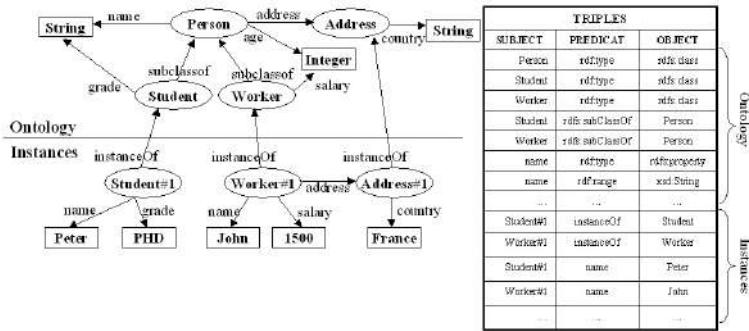


Fig. 1. The vertical table approach

To overcome the drawbacks of the first approach, a dual scheme approach has been proposed. It consists in storing separately ontologies and instance data in two different structures, called ontology and data, respectively [24, 11]. The ontology structure depends upon the ontology model (e.g., RDF Schema, OWL). Figure 2a shows an example of ontology structure for RDF schema. Instances and their properties values are also stored separately. Three different schemes have been proposed to record class belonging [15]. In the two first approaches, each class is mapped onto a table. The third one, called, *hybrid* [2] maps all classes on the same binary table. These schemes are summarized as follows: (1) One table per class with only one column storing all IDs of class instances [2, 14] (see *NOISA* on figure 2b). (2) One table per class with table inheritance using SQL99 capabilities [20] (see *ISA* on figure 2b). (3) A single table with two columns: ID and Class, representing the identifier of an instance, and its ontology class [2], respectively. The ID column may be either the URI or integer identifiers mapped on URI in a particular binary table (called "instances").

Three representation of property values are possible: (1) binary tables, (2) vertical table of triples and (3) hybrid approach consisting of a set of triple

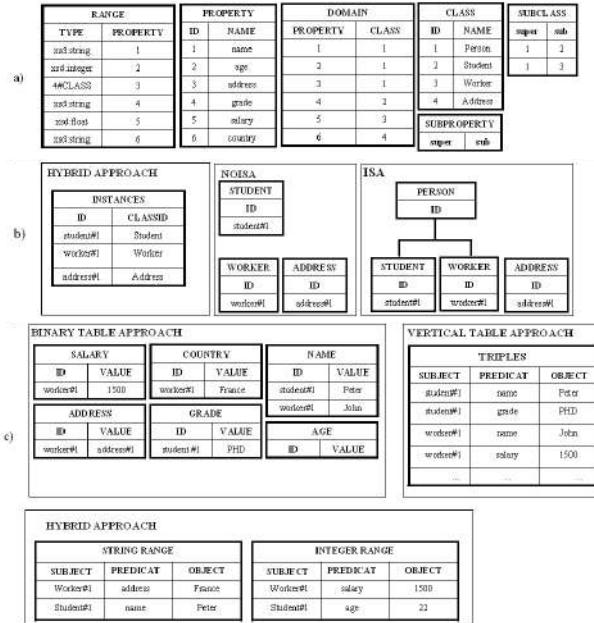


Fig. 2. (a) Ontology Schema in the dual scheme approach. (b) Instance scheme alternative representations. (c) Property value scheme alternative representations.

tables (one per a range data type) (see Figure 2c). It is worth noticing that both the vertical table approach, and the dual scheme approach with hybrid representation of instances and properties involve a small number of large tables when other dual scheme approaches involve a large number of smaller tables.

A number of benchmarks were already proposed to compare the existing approaches [1, 2, 10, 11, 15]. The main findings may be summarized as follows:

- The vertical table approach may only provide similar query results with the dual schema when the vertical table is clustered [4], and/or when materialized views represent the dual scheme content [15]. Even in this case the vertical table approach provides worst results for taxonomic queries, i.e., those queries require subsumption inference [15].
- The dual scheme approach with hybrid representation of instance belonging and property values also requires clustering operations of these tables [15], else this approach is outperformed by dual scheme with unary instance tables and binary property tables.
- The clustering operation is time consuming. An experiment was done on a small database¹ and we got about 3 mins and 30 seconds, which is very significant. This is because a clustering operation has to be performed each

¹ DB_10P_1K: database which has 10 valued properties and 1K instances per class (see section 4.1).

time for each update query. Thus the cost of updating the database schema when unary and binary representations are used is to be compared with the cost of clustering data in vertical table and hybrid approaches.

- Most popular OBDB management systems (e.g., Sesame [1], RDFSuite [2] and DLDB [11]) use the binary representation to store ontologies instances.

Thus, from the previous experiments, we may conclude that the dual scheme approach with unary instance and binary property representations appears as a suitable approach for those databases that are both rather large and often updated. Nevertheless, when the number of properties associated with each instance grows, browsing or querying instances becomes more and more difficult since it requires a large number of joins.

The paper is organized as follows: Section 2 presents the context of our ontology-based database architecture, called OntoDB and the PLIB ontology model. Section 3 presents the architecture of OntoDB and our proposed representation, called, *table per class*, for storing ontology-based data. Section 4 presents our experimental results. Section 5 concludes the paper and presents some perspectives.

2 Context of Our Study

In the 90s, to allow the exchange of electronic catalogues of industrial components, an ontology model for technical domain was developed and published as an international standard known as PLIB (ISO 13584-42: 98). A model to exchange objects described using ontologies was developed [13] and also standardized (ISO 13584-25:2003). In the beginning of 2001, a new project called OntoDB was launched. It aimed to store, exchange, integrate and process industrial catalogues modeled as ontology-based data associated with a formal ontology. PLIB-based ontologies were first targeted. We outline below both the PLIB ontology model and the model for PLIB-based instance data.

2.1 PLIB Ontology Model

The PLIB ontology model is technical domain-oriented as it supports four main capabilities broadly used in engineering: (1) a property value may depend upon its evaluation context (e.g., the length of an axis depends upon its temperature), thus a property may be a function, (2) the property value may be associated with a measure unit (e.g., a temperature may be expressed in degree Celsius), (3) an object must be characterized by one single "characterization class" (associated with properties), but it may also be associated with any number of discipline-specific ontology class, the point of view itself being represented by an ontology class [12], and (4) an object may be classified in any number of "classification class" (not associated with properties). It is worth noticing that this particular taxonomy of meta classes allows PLIB ontology to represent a number of Semantic Web applications such as Web Portal catalogs [2].

PLIB ontologies are domain ontologies: they describe by means of classes and properties all the consensual entities of the target domain. Each property is defined in the context of a class, that constitutes its domain, and it has a meaning only for this class and its possible subclass(es). To avoid the contextual character of a classification, in a PLIB ontology, a class is created only if it is necessary to define the domain of a property that would not be understood in the context of its super-class. Inversely, a property can be defined in the context of a class even if it does not apply to all its instances or subclasses. The single condition is that it is defined in an unambiguous way.

Thus, class hierarchies of PLIB ontology are extremely "flat". They do not define all the possible classes existing in a given domain, but they define only a canonical minimal vocabulary that consists only of primitive concepts. This vocabulary shall only make it possible to describe, in a single way by a class belonging and a set of properties value pairs, all instances which are subject of a common understanding by domain experts. Any entity existing in a domain can thus be described, either directly in terms of the shared ontology or by adding additional classes that refine shared concepts and/or that add properties to the shared ontology.

2.2 PLIB Instance Data Model

Contrary to individuals of description logic-based ontologies that may belong to any number of non connected ontology classes, the PLIB instance model is strongly typed. This means that (1) each instance belongs to exactly one minimal characterization class (called its basis class which is the minimum for subsumption order of all the characterization classes to which the instance belongs), (2) each property is defined in the context of a characterization class that defines its domain of application, and is associated with a range and (3) only properties that are applicable in the context a characterization class may be used for describing its instance. This assumption, rather similar to the OEM model in the TSIMMIS project [5] ensures that there exists an envelop modem that fits with any instance of a class: namely the set of all its applicable properties. But, unlike strongly typed conceptual models, an instance is not required to be associated with values for all the applicable properties of its basis class.

3 OntoDB Model Architecture

We describe below the OBDD architecture we have proposed for storing ontologies and PLIB-instance data. The main objectives of our architecture model are: (1) to support automatic integration and management of heterogeneous populations whose data, schema and ontologies are loaded dynamically, (2) to support evolutions of the used ontologies and of ontology scheme, and (3) to offer data access, at the ontology level, whatever is the type of the used DBMS. Our architecture is composed of four parts. Parts 1 and 2 are traditional parts available in all DBMSs, namely the *data* part that contains instance data and *meta-base*

part that contains the system catalog. Parts 3 (*ontology*) and 4 (*meta-schema*) are specific to our OntoDB (Figure 3). *Ontology* part allows to represent ontologies in the database. Note that each ontology that can be represented and exchanged as models (following Bernstein's terminology) can be supported by our OntoDB model. We can cite for instance, OWL [6], and in particular PLIB. When the target DBMS is relational, the ontology part schema is defined using an object/relational mapping. The *meta-schema* part records the ontology model into a reflexive meta model. For the ontology part, the meta schema part plays the same role as the one played by the meta-base in traditional DBs. Indeed, this part may allow: (1) a generic access to the ontology part, (2) a support of evolution of the used ontology model, and (3) a storage of different ontology models (OWL, PLIB, etc.).

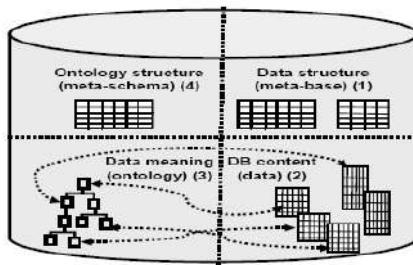


Fig. 3. Our OntoDB Architecture

By means of naming convention, the meta-base part also represents the logical model of the content, and its link with the ontology, thus representing implicitly the conceptual model of data in database relations. Therefore, our OBDB model, called *OntoDB* represents explicitly: (1) ontologies, (2) data scheme, (3) data, (4) the links between the data and their schema and (5) the link between the data and the ontology.

Representation of ontology-based data

Ontologies describes semantic of objects of a given domain. This is done by assigning objects to ontological classes and describing them with ontological properties. According to the used ontology model, various constraints govern such descriptions. For instance, in RDF Schema, an object may belong to any number of classes and can be described by any set of properties. Therefore each domain object has its own structure. Oppositely, a database schema describes "similar" objects by an identical logical structure, in order to speed up queries using indexing schemes. Without any particular assumption, the only possible common structure consists in associating each object with any subset of classes and any subset of properties. All OBDB schemes discussed in section II support this capability.

This approach is not efficient in application domains (like engineering), where each instance is characterized by a significant number of properties (e.g., 50),

and this number is considered smaller than the total number of properties of various class instances (e.g., 1000). Agrawal et al. [1] evaluated this approach and showed its inefficiency.

STUDENT			WORKER				ADDRESS	
OID	name	grade	OID	name	salary	address	OID	country
student#1	Peter	PHD	worker#1	John	1500	address#1		France
...		

Fig. 4. Table per class representation approach

Thanks to the strong typing assumptions presented in section 2.2, we define a relevant schema for any ontology class. It consists of all the class applicable properties used at least by one instance of the class. This schema might contain in some case a number of null values. This scenario is less frequent, since in industrial component catalogues [2] same properties are used by various instances of the same class. Thus our approach supports efficient query processing. Note that this schema definition implies a major difference between object-oriented databases (OODB) and our OBDB. In an OODB, subsumption means inheritance of properties/attributes. All the properties defined in some class do exist in all its subclass(es). The only mechanism for property sharing between two subclasses of a class is to factorize this property at the level of the mother class. But then the property shall appear in all sibling classes. In OBDBs, inheritance is intentional: it concerns only the ontology level. Represented properties may be any subset of applicable properties.

To summarize our ontology-based data representation, we create a table for each class in the database. Its columns consists of a subset of applicable properties those that are used by some of its instances. Figure 4 shows an example of our representation structure with ontology data of Figure 1. In the following section, we call *table per class* our approach of representation of ontology-based data. Note that our OntoDB model records explicitly the structure of each class table and that taxonomies queries (i.e., queries that require subsumption inference) are first evaluated intentionally (to know which classes use each particular property) before querying the data part.

4 Evaluating Instance Representation Schemes

In order to study the effectiveness of our representation of ontology-based instance data, we carried out a series of experiments to compare two representation scheme: unary instance and binary property representations (*binary* for short) vs *table per class*. Our initial intent was to compare also with the *vertical* representation. Finally, we restricted to binary representation. As an experimental platform, we use the ORDBMS PostgreSQL-7.4 (emulated on cygwin) installed on a Pentium 3.7 GHz CPU, 6 GO of RAM, 200 GO of Hard Disk. In all our experiments, a cache memory of 50 MO has been used.

	Serie 1			Serie 2		
	DB_10P_1K	DB_25P_1K	DB_50P_1K	DB_10P_10K	DB_25P_4K	DB_50P_2K
Number of valued properties / class	10	25	50	10	25	50
Number of instances / class	1K	1K	1K	10K	4K	2K
Number of initialised classes	134	134	134	134	134	134
Total number of instances in DB	134K	134K	134K	1340K	536K	268K
Data size in DB	0,341 GO	0,84GO	1,68GO	3,41GO	3,41GO	3,41GO

Fig. 5. The used databases

4.1 Databases

To perform our experiments, we use a real and representative ontology of our application domain. It describes the various kinds of electronic components together with their characteristic properties. This ontology is published as an International Standard in 1998, IEC 61360 [9] and has 190 classes: 134 leaf classes and 56 no leaf classes. These classes have a total of 1026 properties. The average deep of the IEC ontology hierarchy is 5. To facilitate the computation of the sizes of the used databases, all ranges of properties were changed to have a string (255) as their range. A generator of each class population is developed. Various populations were generated, by varying the number of instances and the number of valued properties used by each class. We denote by TP and TC , the *binary* table per property and *table per class* approach, respectively. Let DB_aP_iK be a database with "a" properties and "iK" instances per class. For example, BD_50P_2K is a database with 50 valued properties and 2K instances for each class in the database.

To conduct our experiments, six databases are created (Figure 5). These databases are classified into two series: databases in the first series (Serie1) have the same number of instances per class and a different number of properties: BD_10P_1K , BD_25P_1K and BD_50P_1K . This series allows us to evaluate the effect of database size on query performance. Databases in the second category (Serie2) have the same size ($InstancesNumber \times PropertiesNumber$), but different number of instances and of properties per class: BD_10P_10K , BD_25P_4K and BD_50P_2K . This classification allows us to study the effect on query performance of the number of properties and of instances per class. Note the series 2 databases contain 13.5 millions of RDF triples.

4.2 Query Taxonomy

Three classes of queries are considered: *targeted class queries*, *no targeted class queries* and *update queries*. In a targeted query, a user knows the classes that she wants to query. In a non targeted class of queries, a user does not know the classes that she is looking for. In this study, we consider PSJ queries (projection, selection and join) executed on leaf and non-leaf classes. Note that each query is performed once to warm up the database buffer and then performed at least three times in order to get a mean running time.

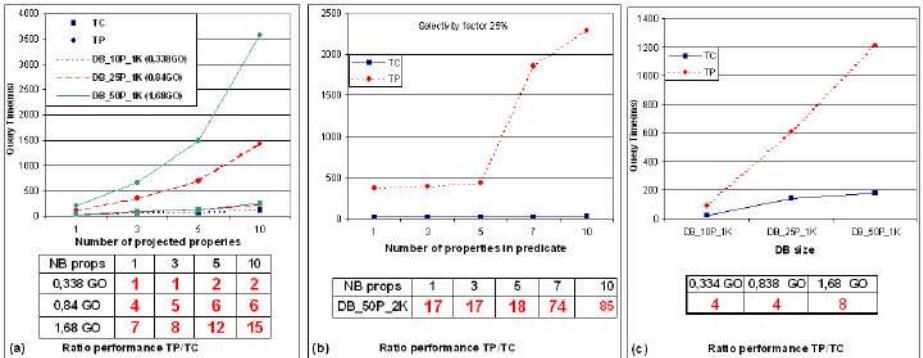


Fig. 6. (a) Projection for various number of properties and various size of databases, (b) Selection with various number of properties (DB_50P_2K)(c) Join within a leaf class for various size of databases

4.3 Performance Results for Targeted Class Queries

We conduct three series of experiments: (1) projection queries, (2) selection queries, and (3) join queries. Figure 6 below shows, for each database, the query execution time as a function of a particular criteria. Table containing a ratio (when present), gives how many times the classical TP representation is slower than our TC representation. Due to space limitations, we are not showing the complete experimented results that are described in [7].

Projection within a leaf class. We performed four queries with 1, 3, 5 and 10 projected properties, respectively. Figure 6a summarizes the execution time. The response time for TC is relatively constant, when the number of projected properties increases, while for TP approach, the variation of the number of projected properties means the augmentation of join operations. Therefore the cost increases dramatically, when all the relevant binary tables cannot be simultaneously fetched in the main memory. For the biggest database of our benchmark (1,7 to 3,4 GO), projection on 10 properties is about *10 to 15 times faster with TC than TP*.

Selection within a leaf class. Figure 6b shows performance of selection queries on one of the biggest database, namely, *DB_50P_2K*. We varied the number of properties in the selection predicate from 1 to 10. Once again, the worst performance is justified by the number of join operations and the sizes of property tables that may cause an important IO overhead. Changing the selectivity factor of the predicate that contains only one attribute does not change significantly the behavior of both representations. Globally, *TC representation outperforms TP by a factor between 17 and 85*.

Join Operations within a leaf class. Figure 6c shows the performance of join queries performed on databases of Series1. The queries return 1 property value per class. The join selectivity is fixed to 0.25%. *TC* approach has better perfor-

mance than *TP*. Variation of databases size increases the ratio between *TP* and *TC*. The reason of worst performance of *TP* is justified by the size of the binary property tables and the fact that a preliminary join is needed between the class table and the property binary tables. In our domain of study, *TC outperforms TP between 4 and 8.*

Projection and selection within non-leave classes. We have also evaluated projection and selection within a non-leaf class with seven subclasses. Query response time in a non-leaf class is the *sum* of queries response time performed in each subclass of the non-leaf class. So, the shape of the curves of performances are identical in queries on leaves and non-leaves classes. In these experiments, *the ratio TP/TC is between 11 and 35.*

4.4 No Targeted Class Queries

When the class to be queried is unknown, the advantages of the *table per class* approach may disappear. Such queries may be formulated as follows: "find all instances in the database that have value val_1 for a property P_1 AND/OR val_2 for a property P_2 ", etc. Execution of this kind of queries in *TC* approach is performed in two steps (1) find all classes in the database using properties (P_1 , P_2) perform selection queries on all found classes. In *TP* approach, execution of *non-targeted queries* are performed directly by joining tables of the properties present in the query predicates. We note that this kind of query is hardly used in our application domain: we never request "an object with the weight equals 1 kilogram". Moreover, if one does not know the class of an object, we need, at least, several properties for characterizing this object. Therefore, such queries request projection on several properties.

We ran these queries against databases of growing sizes (Series 1). We varied the number of projected properties to 1, 3, 5 and 10 to represent realistic queries. *TP approach is more efficient than TC approach as long as queries return less than 5 properties. Beyond this number of properties, TC approach becomes more efficient.* The worst performance of *TC* is identified when a small number of properties is requested (this is due to access time to the *ontology part*). Notice that the time find all classes (step 1) is relatively constant, when we vary the number of properties in the queries, contrary to *TP* approach where every new property causes one more join. So, when the number of requested properties increases, to compute classes in the first step in *TC*, becomes smaller than the time of joins in the *TP* approach.

4.5 Update Queries

Figures 7 shows performance results of insertion and update queries. Queries are ran on databases with different sizes. Both figures show that *TC* is more efficient than *TP*. The worst performance of *TP* for insertion scenario results from the fact that all tables of the valued properties need to be loaded in the main memory, while *TC* loads only one table. For update queries (concerning

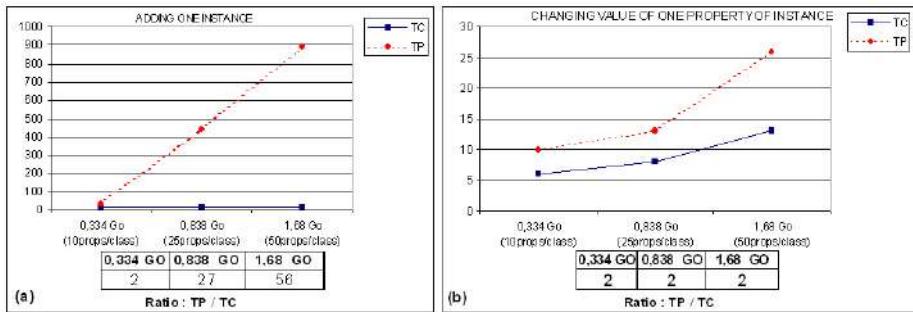


Fig. 7. Insert and Update Queries

only one property value), the worst performance of *TP* is due to the size of the property table that needs to be loaded. The cost ratio between *TP* and *TC* ranges from 2 to 56 for insertion and is about 2 from each update for a single property.

5 Conclusion

A number of ontology-based database structures have been proposed during the last five years. Most of them are targeted to support real scale Semantic Web applications. Several benchmarks were proposed to compare their performance. These benchmarks focus mainly on the class structure and taxonomy queries (i.e., retrieving proper or transitive instances of a particular class or property). Other applications of ontology-based database exist and focus mainly on property-value pairs (case of engineering databases and a number of B2B applications), where an instance data consists of a class belonging and a number of property-value pairs. In this paper, we firstly present an ontology model, secondly an OBDB architecture, and finally, a structure to store instance data, called *table per class*. Our proposed benchmark for comparing this approach with the binary table approach, uses a real standardized ontology with real size databases containing up to 15 millions of RDF triples. Our benchmark is based on three kinds of queries: (1) targeted class queries, (2) non targeted class queries, and (3) insertion and update queries. For queries (1) and (3), the table per class approach outperforms the classical binary table approach with ratio often bigger than 10. The only case, where the binary approach is better is for no targeted class queries, when a user only requests a very small number of property values. Note that this kind of queries nearly never happens in our application domain.

Our OntoDB prototype is already supporting more than millions of instances with dozen of properties, but it mainly uses PLIB ontologies. We are currently working to make its ontology model more flexible to integrate other kind of ontologies. We are also improving the ontology implementation to speed up the ontology browsing process. Finally, we are developing a SQL oriented OBDB query language that integrates most of RQL and of SQL99 capabilities.

References

1. R. Agrawal, A. Soman, and Y. Xu. Storage and querying of e-commerce data. In *Proc. VLDB'01*, pages 149–158, 2001.
2. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. On storing voluminous rdf descriptions: The case of web portal catalogs. In *Proc. of WebDB'01 (co-located with ACM SIGMOD'01)*, 2001.
3. B. McBride. Jena: Implementing the rdf model and syntax specification. In *Proc. of the 2nd Intern. Workshop on the Semantic Web*, 2001.
4. J. Broekstra, A. Kampman, and F.V. Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *Proc. of the First Inter. Semantic Web Conf.*, pages 54–68, 2002.
5. S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, pages 7–18, Marsh 1994.
6. M. Dean and Schreiber. W3C web ontology language reference. *W3C Recommendation (2004)*, February 2004.
7. H. Dehainsala, G. Pierra, and L. Bellatreche. Managing instance data in ontology-based databases. Technical report, LISI-ENSMA,<http://www.lisi.ensma.fr/ftp/pub/documents/reports/2006/2006-LISI-003-DEHAINSALA.pdf>, 2006.
8. S. Harris and N. Gibbins. 3store: Efficient bulk rdf storage. In *Proc. of the 1st Intern. Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, 2003.
9. IEC. Iec 61360 - component data dictionary. *International Electrotechnical Commission*. Available at <http://dom2.iec.ch/iec61360?OpenFrameset>, 2001.
10. L.Ma, Z. Su, Y. Pan, L. Zhang, and T. Liu. Rstar: an rdf storage and query system for enterprise resource management. *thirteenth ACM international conference on Information and knowledge management*, 2004:484 – 491.
11. Z. Pan and J. Heflin. Dldb: Extending relational databases to support semantic web queries. *ISWC'2003*, 2003.
12. G. Pierra. A multiple perspective object oriented model for engineering design. In *New Advances in Computer Aided Design & Comp. Graphics*, pages 368–373, 1993.
13. G. Pierra. Context-explication in conceptual ontologies: Plib ontologies and their use for industrial data. *to appear in Journal of Advanced Manufacturing Systems, World Scientific Publishing Company*, available at <http://www.lisi.ensma.fr/ftp/pub/documents/papers/2006/2006-JAMS-Pierra.pdf> 2006.
14. K. Stoffel, M.G. Taylor, and J.A. Hendler. Efficient management of very large ontologies. In *Proc. of American Association for Artificial Intelligence Conference (AAAI'97)*, 1997.
15. V. Christophides Y. Theoharis and G. Karvounarakis. Benchmarking database representations of rdf/s stores. In *Fourth International Semantic Web Conference (ISWC'05)*, November 2005.

Continuously Maintaining Sliding Window Skylines in a Sensor Network

Junchang Xin¹, Guoren Wang¹, Lei Chen², Xiaoyi Zhang¹, and Zhenhua Wang¹

¹ Institute of Computer System, Northeastern University, Shenyang, China
wanggr@mail.neu.edu.cn

² Department of Computer Science and Engineering, Hong Kong University of
Science and Technology, Hong Kong, China
leichen@cs.ust.hk

Abstract. Currently, wireless sensor network has been widely used in environment monitoring. The skyline query, as an important operator for multiple criteria decision making and data mining, plays an important role in many sensing applications. Though skyline queries have been well-studied in traditional database system, the existing solutions designed for data stored in a centralized site are not directly applicable to sensor environment due to the unique characteristics of wireless sensor network. In this paper, we propose an energy-efficient algorithm, called Sliding Window Skyline Monitoring Algorithm (SWSMA), to continuously maintain sliding window skylines over a wireless sensor network. Specifically, SWSMA employs two types of filters within each sensor to reduce the amount of data transferred and save the energy consumption as a consequence. In addition to SWSMA, a set of optimization mechanisms are also discussed to improve the performance of SWSMA. Our extensive simulation studies show that SWSMA together with the optimization techniques performs effectively on reducing communication cost and saving the energy on monitoring sliding window skylines.

1 Introduction

In recent years, wireless sensor networks (WSN) have been widely used in environmental monitoring [16,24], such as earthquake monitoring, habitat monitoring, agriculture monitoring, coal mine environment monitoring, etc. Current sensors are generally cheap, resource-constraint and battery powered, it is not possible or at least very difficult to change batteries. Therefore, applications over sensor networks need a scalable, energy-efficient and fault-tolerant method to monitor the tremendous data generated by sensors. Among all the queries, the skyline query, as an important operator for multiple criteria decision making and data mining, plays an important role in many sensing applications.

A skyline query is defined as following:

Definition 1. Assume that we have a relational database, given a set of tuples T , a skyline query retrieves tuples in T that are not dominated by any other tuple. For two tuples t_i and t_j in T , tuple t_i dominates tuple t_j if it is no worse than t_j in all dimensions and better than t_j in at least one.

Within a sensor network environment, data are collected by each sensor node periodically. It is impossible or meaningless to conduct skyline queries over the infinite data streams collected by sensors. Thus, *sliding window skylines*, which seek the skylines over the latest data that are constrained by a sliding window, are very useful for some data monitoring applications. For example, an ornithologist who has been studying birds in the forest may want to know when and where certain kinds of birds are more likely to be discovered. Existing solutions for skylines can not be applied to the sensor environment directly due to distributed nature of the sensory data. In addition to that, as we mentioned, energy is the precious resource in the sensor network and wireless communication is the main consumer, therefore, the sliding window skylines over a wireless sensor network raise up a new challenge on how to minimize the communication cost, which is not addressed by the centralized skyline solutions.

In this paper, we propose an energy-efficient algorithm, called Sliding Window Skyline Monitoring Algorithm (SWSMA), to continuously maintain sliding window skylines over a wireless sensor network. SWSMA employs two types of filters within each sensor to reduce the amount of data transferred and save the energy consumption as a consequence. The contributions of this paper are:

1. We prove theoretically that skyline queries are decomposable, which indicate that in-network computation can be applied to skylines;
2. We propose an energy efficient Sliding Window Skyline Monitoring Algorithm (SWSMA) to continuously maintain sliding window skylines by employing two types of filters to avoid transmission unqualified tuples;
3. In addition to SWSMA, a set of optimization mechanisms are also discussed to improve the performance of SWSMA.

2 Related Work

There are various query process models having been proposed for sensor networks, in which TinyDB [13] [14] [15] and COUGAR [25] are the two typical systems. Both of them provide a SQL-Like interface to implement aggregation operators, such as MAX, MIN, AVERAGE, SUM, and COUNT. The skyline query was first investigated in [3], where several methods were presented, including SQL implementaton, divide-and-conquer (DC) and block-nested-loop (BNL). Chomicki et al. [5] present a pre-sort method, which sorts the dataset according to a monotone preference function and then computes the skyline in another pass over the sorted list. Two progressive methods, Bitmap and Index, are presented in [22]. Since the nearest neighbor (NN) is sure to belong to the last skyline, Kossmann et al. [10] present a progressive on-line method based on NN, which allows user to interact with the process. Papadias et al. [19] use R-tree to further improve the performance of algorithm presented in [10].

The methods above are all based on centralized scenarios. So far, we do not find any approach having been proposed to address skyline queries over a sensor network. The most related works to ours are some studies about skylines in a distributed scenario. Balke et al. [4] extend the skyline problem to the world wide web in which the attributes of an object are distributed in different

web-accessible servers and presented a basic distributed skyline algorithm (BDS) and an improved distributed skyline algorithm (IDS) that compute the skyline in such a distributed environment. BDS uses a simple method to identify a subset of the objects that includes the skyline, and then filters away all the non-skyline objects in that subset. IDS finds the subset more quickly than BDS with a heuristic approach. Later, Lo et al. [11] propose a progressive distributed skyline algorithm (PDS), based on progressiveness and rank estimation, to improve the performance of BDS and PDS. Huang et al. [9] propose a filtration policy to reduce communication cost among mobile devices and a hybrid storage model to reduce the execution time on each single mobile device, which is similar to our proposal. However, their approach mainly focuses on answering skyline queries on one timestamp, i.e. snapshot skylines, ours focuses on continuously monitoring sliding window skyline queries.

There are some works having been proposed to answer sliding window skyline queries with the focus on handling the characteristics of stream data. Tao et al. [23] propose a framework to continuously monitor skyline over stream data. Lin et al. [12] explore the problem of n -of- N skyline queries computing skyline against any most recent n elements in the set of the most recent N elements and present a pruning technique to reduce data capacity, an encoding scheme to reduce memory space, and a new trigger based technique to continuously process an n -of- N skyline query.

3 Preliminaries

In this section, sensor stream is first introduced, followed by sliding window skylines, and finally, some related properties which are the foundation of our filtering algorithms are presented.

3.1 Sliding Window Skyline

In a sensor network, data are periodically collected by sensor nodes, therefore, strictly speaking, sensory data collected by a sensor network cannot be simply considered as a traditional database. It is more like a distributed, multiple data stream system in which all sensor streams are append-only.

Since the data stream is infinite, and the volume of a complete stream is theoretically boundless, it is impossible to carry out skyline operation after all data have been collected. In this paper, *sliding window skyline* is considered. Sliding window skyline only considers the latest set of data. Each tuple in data stream has a timestamp $t.arr$ indicating its arrival time. If the size of sliding window is set to W , and lifespan of the tuple is $[t.arr, t.exp]$, we have $t.exp = t.arr + W$. All the valid data in the interval $[t.curr - W, t.curr]$ of the sliding window will be used to compute skylines when the current time is $t.curr$.

3.2 Properties

We denote the whole tuple set in a sensor network as T , the tuple set for each node as T_i . Furthermore, we assume that the dimension set of the tuple set is

D , dimensionality of tuple set is $|D|$, and n is the number of sensor nodes. We use SKY to stand for skyline operator, \succeq for the dominance relationship, and $t.x_d$ for the d^{th} attribute of tuple t .

Based on the definition of skylines in Section 1, we have the following Lemma.

Lemma 1. *Let t_i , t_j and t_k be three tuples in T . If $t_i \succeq t_j$ and $t_j \succeq t_k$, then $t_i \succeq t_k$.*

Proof: According to the definition of dominance relationship,

$$\therefore \forall d \in D, \quad t_i.x_d \geq t_j.x_d \wedge t_j.x_d \geq t_k.x_d \quad (1)$$

$$\exists d \in D, \quad t_i.x_d > t_k.x_d \quad (2)$$

$$\therefore (\forall d \in D, \quad t_i.x_d \geq t_k.x_d) \wedge (\exists d \in D, \quad t_i.x_d > t_k.x_d) \quad (3)$$

So we can conclude $t_i \succeq t_k$. \square

If an operation is decomposable, it can be computed in-network [14], which means the computation can be carried on within each sensor and the intermedin results will be transmitted during the query data collection phrase. Thus, many redundant transmissions can be saved. Fortunately, skyline in sensor network has this attractive property.

Theorem 1. *The skyline query in sensor network is decomposable.*

Proof: According to Lemma 1 and the definition of skyline query,

$$SKY(T) \subseteq \bigcup_{i=1}^n SKY(T_i) \quad (4)$$

$$SKY(T) \subseteq SKY\left(\bigcup_{i=1}^n SKY(T_i)\right) \subseteq T \quad (5)$$

$$SKY(T) = SKY\left(\bigcup_{i=1}^n SKY(T_i)\right) \quad (6)$$

It satisfies the formula $f(v_1, v_2, \dots, v_n) = g(f(v_1, v_2, \dots, v_k), f(v_{k+1}, \dots, v_n))$ given in [6]. So skyline query in sensor network is decomposable. \square

Theorem 1 indicates that in-network computation can be applied to skylines. In order to further reduce the tuples transmitted among the sensors, setting a filter within each sensor becomes essential. The following Theorem implies the effect of using a tuple as a filter.

Theorem 2. *If tuple t , whether exists or not, is dominated by a valid tuple, all tuples dominated by t will not belong to skyline. Here valid tuple means the tuple that does exist and is unexpired.*

Proof: Immediate deduct from lemma 1. \square

4 Sliding Window Skyline Monitoring Algorithm

In this section, we present the Sliding Window Skyline Monitoring Algorithm (SWSMA), specifically, the computation and maintenance modules are discussed in Section 4.1 and 4.2, respectively.

4.1 The Computation Module

In this paper, we design our skyline algorithms based on the popular data routing structure, tree-based routing structure, which is described in TAG [14,13] and Cougar [25]. In tree-based routing structure, a spanning tree is created with the base station as the root. To compute a skyline, the naive approach is a centralized one, i.e., collecting all the data through the tree-based routing structure to the base station and compute the slide window skylines at the base station. This will cause a large amount of communication cost and network congestion. Therefore, this method is unpractical for wireless sensor network. The possible improvement methods are listed as following.

Merge Approach. From Theorem 1, we learn that skyline operation is decomposable, thus, one feasible method is to use in-network computation. That is to compute skyline in-network whenever possible, the intermediate node merges its own skyline and the skyline results sent by its children, then sends the merged result to its parent. Most of the tuples that belong to local skyline and not global skyline are filtered out on the intermediate nodes. However, there are still many tuples that do not belong to the final skyline having been transmitted.

Tuple Filter Approach. If a tuple belongs to local skyline, not to global skyline, there must exist a global skyline tuple dominating it. The tuple does not need to be transmitted, if it is known in advance that there is a global skyline tuple dominating it. The transmitted data size will be greatly reduced when the tuple which dominates the most tuples is found and informed to other nodes.

To be general, suppose tuple set T is in $|D|$ -dimensional space, and the i^{th} dimension range is $[0, U_i]$. X and Y denote tuples in T . $(x_1, x_2, \dots, x_{|D|})$ and $(y_1, y_2, \dots, y_{|D|})$ are the corresponding coordinates of X and Y . If the probability density function of the tuple in T is $p(X) = p(x_1, x_2, \dots, x_{|D|})$. The total amount of tuples that tuple $Y(y_1, y_2, \dots, y_{|D|})$ can dominate is

$$c(Y) = |T| \times \int_R p(x_1, x_2, \dots, x_{|D|}) dx_1 dx_2 \dots dx_{|D|} \quad (7)$$

where $R = \{X | x_1 \geq y_1, x_2 \geq y_2, \dots, x_{|D|} \geq y_{|D|}\}$.

The tuple that can dominate the most tuples is the one which has the biggest value according to equation 7. Although $p(X)$ is unknown in most cases, the approximate distribution of $p(X)$ can be easily obtained by random sampling or collecting histograms [7,21]. To reduce the cost of integration, we use a polynomial function to present $p(x)$. After the integration is finished at the base station, we only send a few coefficients of $p(x)$, which enable each node to reconstruct the expression approximately.

Equation 7 is then used in finding the “powerful” filter tuple. After obtaining $p(X)$, steps of computing skyline are as follows.

1. Calculate the value c (according to equation 7) of the stored tuples locally.
2. Find the tuple with the maximum value c using the method of in-network aggregate, and set it as the tuple filter.
3. Broadcast tuple filter to the entire network.
4. Filter out tuples that are dominated by the filter in sensor nodes.
5. Use merge approach to carry out skyline computation.

Grid Filter Approach. Intuitively, for some distribution, tuple filter will dominate most non-skyline tuples with an obvious filtration effect; for other distributions, it will only dominate a part of non-skyline tuple with inferior effect. In order to solve this problem, grid filter approach is introduced.

In grid filter approach, a regular grid is used to partition the data space. Each dimension is divided into s segments, and the extent of each segment is U_i/s . Totally there are s^d cells. A cell.sta is used to record the state. If any tuple falls in the cell, cell.sta is set to 1, otherwise 0. Another option to fill the cell is to pre-process the grid, in which *cell.stas* of the dominated cells are set to 0, meaning that the tuples in the cell are not belong to skyline; *cell.stas* of all the other cells are set to 1, meaning the tuples in the cell may belong to skyline. We call the former one *original grid* and the latter one *pre-processed grid* for distinction. To determinate whether a tuple is dominated by a grid, the former needs examine all its bottom left cells’ *cell.stas*, while the later just needs to examine its own *cell.sta*. Furthermore, to deal with the merge of grids on intermediate nodes, we operate the “or” on the original grid, while operate “and” on the pre-processed grid. The original grid costs more during the determination and their cost of merge are the same. We conclude that the pre-process grid performs better totally. So we are apt to using the pre-process grid, and the grid mentioned in the following discussion is the pre-processed grid.

Adaptive Filter Approach. Since tuple filter and grid filter have their own pros and cons, tuple filter is more effective on independent and correlated distributions whereas grid filter performs better on anti-correlated ones, a feasible method is to use a selection mechanism to choose the “right” filter to avoid their disadvantages and fully utilize the advantages, and we call this approach adaptive filter approach.

First of all, samples or histograms [7,21] are used to gain the rough distribution of data, and then the adaptive filter approach determines the filter strategy according to specific distribution. If data is approximate independent or correlated distribution, tuple filter will be used in filtration; if data is approximate anti-correlated distribution, grid filter is used in filtration. In this way, merits of both filter strategies can be fully utilized to optimize the system performance. To adapt to the variation of data distribution, base station needs to select a new filter that is suitable for the new data distribution according to current results of skylines, and determines whether or not to broadcast to the whole network based on the computation result of cost-benefit model.

4.2 The Maintenance Module

After computing the initial skyline, new tuple is generated by sensor node, while the old one will be moved out of the window and become expired. A simple and direct way is to use the method presented in Section 4.1 to recompute the skyline periodically, so as to maintain the coherence of global skyline. Obviously this kind of method is unpractical, because there is a great intersection between the old window and the new one. If the overlapped information can be used, data transmission cost in the maintenance phase will be reduced. An effective way should be “update-only”, which means only those new local skyline tuples are transmitted, and those that have been transmitted do not need to be retransmitted. Thus, the communication cost is reduced.

For Merge Approach, there is no extra process needed, while for the two filter-based approaches, how to maintain filters incrementally during global skyline maintenance becomes a critical problem.

Maintenance of Tuple Filter. It is known from theorem 2 that even if the filter tuple is expired, it is not necessary to replace it, for the reason that if there is a tuple in skyline that can dominate it, the filtered tuples will definitely not belong to skyline. There will be no false negative. However, once a tuple f with small $c(f)$ (according to equation 7) is chosen as a filter, the probability that it is dominated by the skyline tuple is high. Thus, it may not be replaced for a long time. The inferior filtration ability will cause a lot of false alarms and a great waste of energy. The replacement of filter can improve the filtration ability, but will cost certain communication cost to replace the old one. There is a tradeoff between benefit and cost, thus, we introduce a benefit-cost model.

$$\text{benefit}(f) = (c(f) - c(f_{old})) \times n \times \bar{t}_f \quad (8)$$

where \bar{t}_f is the average lifetime of filter tuple, and benefit is the increased number of tuples that will be dominated by f compared with f_{old} .

$$\text{cost}(f) = \text{broadcast}(f) \quad (9)$$

The detailed replacement policy of the filter is described as follows:

1. Filter is expired and there is no tuple in skyline that can dominate it.
2. There is a new tuple whose benefit exceeds the cost.

If one of the above is satisfied, the replacement of filter is carried out, and a new tuple will be chosen as the new filter and broadcasted to the network.

Maintenance of Grid Filter. Before discussing the replacement policy of grid filter, the definition of dominate relationship between grid is brought forward.

Definition 2. For two grids g_1 and g_2 , the set of cells whose cell.stas are 0 in g_1 denotes s_1 , and the set of cells whose cell.stas are 0 in g_2 denotes s_2 , if s_1 is the subset of s_2 , we say grid g_2 dominates g_1 .

Theorem 3. If certain grid g is dominated by a valid grid, whether g expires or not, all tuples dominated by g will not belong to skyline, where valid grid is the one that is gained by unexpired skyline tuples.

Proof: Immediate deduct from Definition 2 \square

\square

In the same way, an expired grid does not necessarily need to be replaced. When a better grid appears, a selection should be made on whether replacing or remaining the old one. The standard to evaluate the filtration ability of grid is different from equation 7. The precondition is same as that of tuple filter, then the filtration ability equation of grid is

$$c(Grid) = |T| \times \int_G p(x_1, x_2, \dots, x_n) dx_1 dx_2 \dots dx_n \quad (10)$$

where $G = \{X | X \in cell \wedge cell.sta = 0\}$.

The benefit-cost model of tuple filter can also be applied to grid filter. The replacement policy of grid filter is similar to that of tuple filter.

5 Optimizations

In this section, several optimization approaches are discussed to further improve the performance of SWSMA. The snooping method is applicable to all filter strategies, while the shearing and compressing methods work on the grid filter.

5.1 Snooping

The snooping method aims to use the information of non-child nodes to reduce the communication cost. In snooping mode, the intermediate node not only keeps data sent by its child node, but also snoops messages sent by other nodes. The snooped data is used when computing skyline just as the tuple filter. The data only participates in filtration, and does not enter the final skyline. Some skyline tuples that were meant to be transmitted do not necessarily need to be transmitted to the parent node, since they are dominated by the snooped tuple, which will further reduce data transmission capacity in sensor network.

5.2 Shearing

The shearing method aims to transmit only the part of useful information during the transmission of grid, while the part to be deduced will not be transmitted.

Take skyline operation using min for example. Since the top right edge of grid does not dominate any cell when merging grid, it does not necessarily need to be transmitted. When grid is broadcast as filter, there is no need to transmit its bottom left edge, because $cell.sta$ of each cell of the bottom left edge is always 1. Therefore, the edge of grid can be cut according to different situations, so as to reduce communication cost.

5.3 Compressing

The compressing method aims to reduce the communication cost. For binary string, special compression mechanism can be utilized.

Since the probability of *cell.sta* of each cell being 0 is computable, it is more likely to have successive 1s or 0s in the sequence if cells in grid are sorted by probability. Thus, better compression effectiveness is gained. Encoded mode in [18] is used to compress data to 30% of the original size.

6 Simulation Evaluation

In this section, we mainly compare the performances of merge approach (MA), tuple filter approach (TF), grid filter approach (GF) and adaptive filter approach(AF) based on tree-based structure under the two data distributions, independent and anti-correlated, which are common benchmarks for skyline query [3,23]. The experimental data distribute evenly on 600-1000 nodes with the communication radius of $2\sqrt{2}$ times the area side length occupied by the node itself. The dimension of sensory data ranges from 2 to 4 and the size of sliding window varies from 100 to 500. For each timestamp, each node generates a new tuple, thus there will be n new tuples generated in the whole network. All simulations are completed on pentium 2.8 GHz CPU with 512MByte memory.

In the simulation, the default values number of nodes $n=1000$, cardinality $C=300$ and dimension $d=3$. Since under independent distribution, TF algorithm is adopted by AF, performances of MA, GF and AF will be compared; while under anti-correlated distribution, GF algorithm is adopted, performances of MA, TF and AF will be compared.

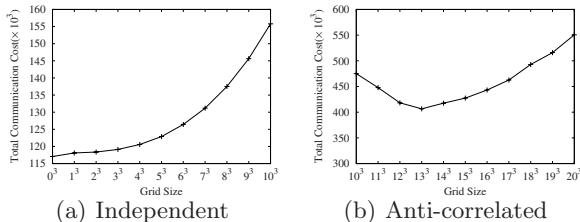


Fig. 1. Effect of GF granularity in computation module

Before performances of each algorithm in computation module are compared, we study effect by grid granularity first. Figure 11 shows the total communication cost (number of bytes of the messages transmitted) under different grid granularity. The lowest grid granularity for independent distribution is 0, and the best grid granularity for anti-correlated distribution is 13. This is because grid will filter out some of the data, at the same time, its computation and broadcasting are not free. For independent data, the broadcast cost acceleration of grid exceeds the effect resulted from filtration with the increase of granularity. While for

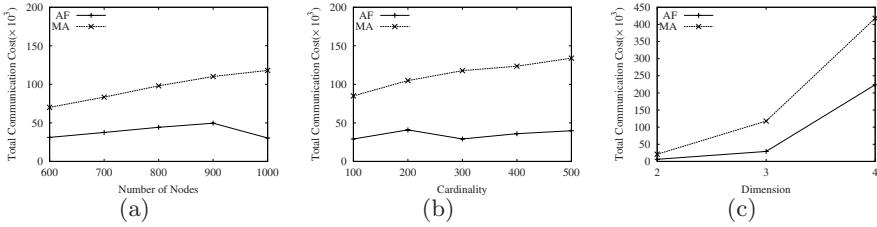


Fig. 2. Performance with independent data

anti-correlated data, cost and benefit balance well. Therefore, in the experiment of computation module, the grid granularity for independent data is 0, which degenerates into MA; while the grid granularity for anti-correlated data is 13.

Figure 2 and 3 present the influence on performance by dimension, cardinality and the number of nodes under independent and anti-correlated data distribution, respectively. It shows that AF is the best under all circumstances. This is because TF can filter a great amount of tuples with far less transmission cost than GF under independent distribution; while GF can filter out several times of data than TF which far exceeds its own transmission cost under anti-correlated distribution. Since AF always chooses the best strategy, its performance turns out to be the best. Meanwhile, cost increases with the increase of dimensions, since the skyline result will increase with a high dimension which leads to the increment of communication cost. Change of cardinality will not directly affect the cost, because there is no obvious functional relationship between the size of the result set and cardinality. For the same reason as the former, change in the number of nodes does not directly affect the cost.

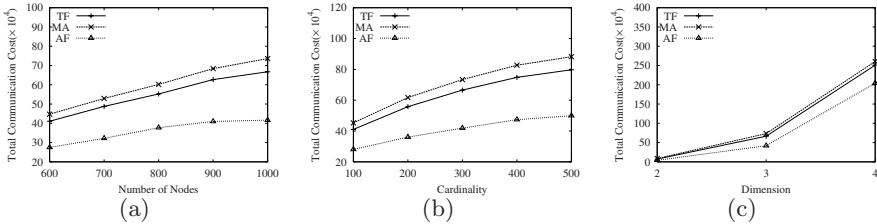
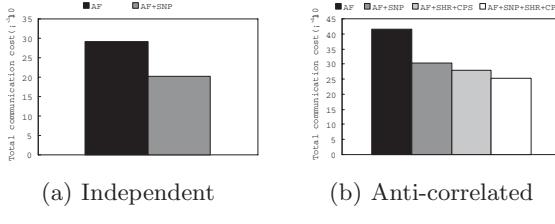


Fig. 3. Performance with anti-correlated data

Figure 4 reports the influence on algorithm performance brought by optimization strategies for independent and anti-correlated distributions, respectively. It shows that snooping (SNP) remarkably improves the efficiency of the algorithm under all circumstances. Because the shearing and compressing methods only apply to grid filter, they just appear in anti-correlated distributions. Figure 4(b) shows that both shearing (SHE) and compressing (CPS) are effective for anti-correlated distributions.

**Fig. 4.** Optimizations in computation module

Next, we study the performances of algorithms in skyline maintenance module. Figure 5 gives the influence on GF performance by grid granularity. Since grid has a long aging in the process of maintenance and computation and broadcast cost will be shared by each time segment, the optimum grid granularity changes. For two different routing structures, the best grid granularity for independent distribution is 25, and it is 15 for anti-correlated distribution.

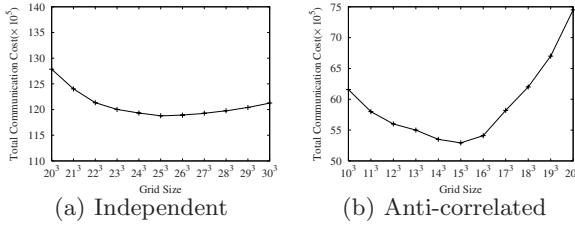
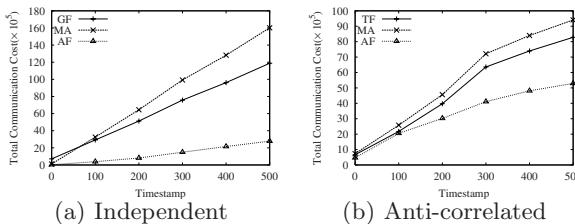
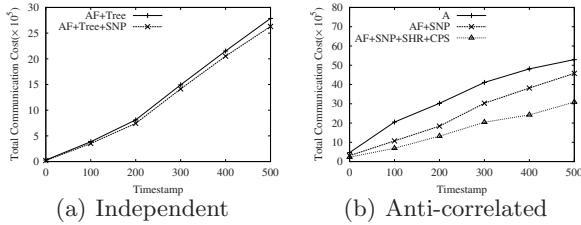
**Fig. 5.** Effect of GF granularity in maintenance module

Figure 6 presents the time-varying regularity of communication cost in maintenance for each algorithm. We can observe that the communication cost increases smoothly with time. Similar to the computation module, AF is the best under independent and anti-correlated distribution. It basically shares the same reason with computation module.

**Fig. 6.** Communication cost varying time

Finally, we study the influence on algorithm performance brought by optimization strategies in maintenance module. Figure 7(a) shows that SNP further

**Fig. 7.** Optimizations in maintenance module

improves performance and reduces communication cost for independent distributions. Figure 7(b) shows that all optimizations work well for anti-correlated distributions.

7 Conclusions

In most sensor network, energy is a critical resource, and is mainly consumed by communication. How to minimize the communication cost for applications in sensor network becomes an essential problem. In this paper, we introduce an energy-efficient algorithm called SWSMA to maintain the sliding window skyline of sensor network. First, merge, tuple filter, grid filter and adaptive filter are proposed to calculate the initial skyline in sensor network, then methods for maintaining filter in continuous query are discussed. Furthermore, some optimizations to improve SWSMA capacity are also presented. The experiment result proves that SWSMA is an efficient method for calculating and maintaining skyline in sensor networks.

Acknowledgement. This work is partially supported by National Natural Science Foundation of China under grant No. 60573089 and 60473074 and supported by Natural Science Foundation of Liaoning Province under grant no. 20052031.

References

1. D. J. Abadi, S.I Madden, and W. Lindner: REED: Robust, Efficient Filtering and Event Detection in Sensor Networks. In *Proc. of VLDB*, 2005.
2. Boris Jan Bonfils and Philippe Bonnet: Adaptive and Decentralized Operator Placement for In-Network Query Processing. In *Proc. of IPSN*, 2003.
3. S. Borzonyi, D. Kossmann, and K. Stocker: The skyline operator. In *Proc. of ICDE*, 2001.
4. W.-T. Balke, U. Guntzer, J. X. Zheng: Efficient distributed skylining for web information systems. In *Proc. of EDBT*, 2004.
5. J. Chomicki, P. Godfrey, J. Gryz, and D. Liang: Skyline with presorting. In *Proc. of ICDE*, 2003.
6. J. Considine, F. Li, G. Kollios, and J. Byers: Approximate aggregation techniques for sensor databases. In *Proc. of ICDE*, 2004.

7. Surajit Chaudhuri, Nilesh N. Dalvi, Raghav Kaushik: Robust Cardinality and Cost Estimation for Skyline Operator. In *Proc. of ICDE*, 2006.
8. Vishal Chowdhary, Himanshu Gupta: Communication-Efficient Implementation of Join in Sensor Networks. In *Proc. of DASFAA*, 2005.
9. Zhiyong Huang, Christian S. Jensen, Hua Lu, Beng Chin Ooi1: Skyline Queries Against Mobile Lightweight Devices in MANETs. In *Proc. of ICDE*, 2006.
10. D. Kossmann, F. Ramsak, S. Rost: Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proc. of VLDB*, 2002.
11. Eric Lo, Kevin Ip, King-Ip Lin, David Cheung: Progressive Skylining over Web-Accessible Database. *DKE*, 57(2): 122-147, 2006.
12. Xuemin Lin, Yidong Yuan, Wei Wang, Hongjun Lu: Stabbing the Sky: Efficient Skyline Computation over Sliding Windows. In *Proc. of ICDE*, 2005.
13. S. Madden, M. Franklin, J. Hellerstein, and W. Hong: The design of an acquisitional query processor for sensor networks. In *Proc. of SIGMOD*, 2003.
14. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proc. of OSDI*, 2002.
15. S. Madden et al.: Supporting aggregate queries over ad-hoc wireless sensor networks. In *Proc. of WMCSA*, 2002.
16. R. Oliver, K. Smettem, M. Kranz, K. Mayer: A Reactive Soil Moisture Sensor Network: Design and Field Evaluation. *JDSN*, 1: 149-162, 2005.
17. Aditi Pandit, Himanshu Gupta: Communication-Efficient Implementation of Range-Joins in Sensor Networks. In *Proc. of DASFAA*, 2006.
18. C. Palmer, P. Gibbons, and C. Faloutsos: ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs. In *Proc. of SIGKDD*, 2002.
19. D. Papadias, Y. Tao, G. Fu, et.al.: An Optimal and Progressive Algorithm for Skyline Querie. In *Proc. of SIGMOD*, 2003.
20. G. Pottie and W. Kaiser: Wireless integrated sensor networks. *Communications of the ACM*, 2000.
21. Bernard W Silverman: Density Estimation for Statistics and Data Analysis. CRC Press, 1986.
22. K.-L. Tan, P.-K. Eng, and B. C. Ooi: Efficient progressive skyline computation. In *Proc. Of VLDB*, 2001.
23. Yufei Tao, Dimitris Papadias: Maintaining Sliding Window Skylines on Data Streams. *TKDE*, 18(3): 377-391, 2006.
24. W. Xue, Q. Luo, L. Chen, and Y. Liu: Contour Map Matching For Event Detection in Sensor Networks. In *Proc. of SIGMOD*, 2006.
25. Y. Yao and Johannes Gehrke: The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3), 2002.

Bayesian Reasoning for Sensor Group-Queries and Diagnosis

Ankur Jain¹, Edward Y. Chang², and Yuan-Fang Wang³

¹ Computer Science UC Santa Barbara, CA 93106
ankurj@cs.ucsb.edu

² Electrical and Computer Engg. UC Santa Barbara, CA 93106
echang@ece.ucsb.edu

³ Computer Science, UC Santa Barbara, CA 93106
yfwang.cs.ucsb.edu

Abstract. As large-scale sensor networks are being deployed with the objective of collecting quality data to support user queries and decision-making, the role of a scalable query model becomes increasingly critical. An effective query model should scale well with large network deployments and address user queries at specified confidence while maximizing sensor resource conservation. In this paper, we propose a *group-query* processing scheme using Bayesian Networks (BNs). When multiple sensors are queried, the queries can be processed collectively as a single group-query that exploits inter-attribute dependencies for deriving cost-effective query plans. We show that by taking advantage of the *Markov-blanket* property of BNs, we can generate resource-conserving group-query plans, and also address a new class of *diagnostic queries*. Through empirical studies on synthetic and real-world datasets, we show the effectiveness of our scheme over existing correlation-based models.

1 Introduction

Sensor network research is strategically positioned at the exciting confluence of sensing, computation, and communication. A sensor network can employ numerous small, inexpensive sensors of the same or different modalities (e.g., biological, chemical, mechanical, and electrical) to perform many useful tasks such as collecting seismic data, monitoring environment, measuring traffic flows, and safeguarding security, to name just a few. These sensors must be carefully managed to achieve two performance goals: 1) conserving power for prolonging useful life, and 2) collecting reliable data for supporting user queries, despite transient noise, sensor failures, and malicious attacks.

A query engine for a typical sensor network supports multiple users, where each user may query for probabilistic estimates of one or more sensor attribute values with some specified confidence level. Recent research efforts have shown that correlations are prevalent between sensor attributes, and queries on costly sensors can be answered efficiently by acquiring data from cheap sensors [98]. In this paper, we advance the traditional correlation model in two directions. First, we employ Bayesian Networks (BNs) for characterizing sensor networks. BNs provide a compact representation of dependencies and offer effective inferencing methodologies. Such a model captures both

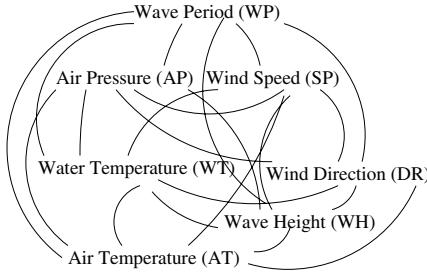


Fig. 1. Correlation model for NDBC data

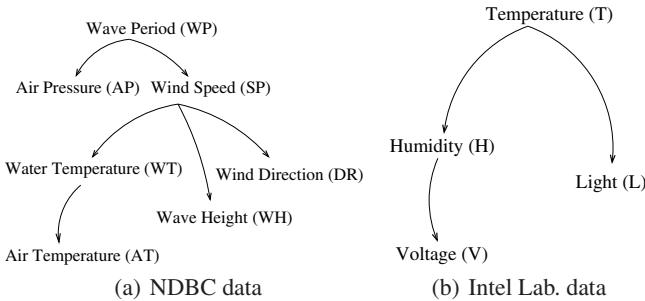


Fig. 2. Compact representation using BN

the stochastic characteristics of, and the statistical relations between, sensor attributes. The use of BNs can provide efficient query-plan generation for traditional aggregate queries, as well as for diagnostic queries. Second, we consider the problem of *group-query processing*. When multiple queries are issued, instead of processing each query individually, we exploit inter-query relations to further reduce the overall resource usage.

To illustrate the advantages of using the BN over the traditional correlation model [98], we present two simple examples generated using data from the National Data Buoy Center (NDBC) [25] and the Intel Lab [3], respectively. Figure 1 shows that a correlation model must maintain all pairwise correlations. In contrast, Figure 2(a) shows that the BN provides a much more compact representation of essential relations between sensor nodes. In general, for n attributes, with an average in-degree of d , the number of probability values required to represent the joint probability for a BN is $O(nd)$, in contrast to $O(n^2)$ required by the correlation model. Since $d \ll n$ for typical real-world sensor networks, the BN provides a much more succinct representation of a sensor network.

The work of [98] successfully points out that a query on an expensive sensor can be answered using other inexpensive and statistically correlated sensors. When n is large, however, generating a good plan using such a correlation model can be time-consuming. The BN model can reduce the search space of a node (representing a sensor query) from $O(n)$ to the nodes in its *Markov blanket* [24], which consists only of the node's immediate parents, its immediate children, and other parents of its children. Let us revisit the example in Figure 2(a). Suppose a user queries "water temperature." Its Markov

blanket tells us that an alternate plan should consider only “air temperature” or “wind speed,” as they have the most direct and significant influence on “water temperature,” instead of all attributes as shown in Figure 1. Moreover, a correlation model [9,8] could make suboptimal choices in the plan. For example, in Figure 2(b) even though “voltage” is highly correlated with “temperature,” “voltage” is conditionally independent of “temperature” given “humidity.” Hence, given the value of “humidity”, “temperature” has no effect on “voltage.” More importantly, “humidity” could be a cheaper source to query than “temperature.” The model of [9,8] would miss this better choice in its query-plan generation. Thanks to the *Markov blanket* property of the BN, our model can not only efficiently reduce the search space for generating a query plan, but also generate a more effective plan. We can take further advantage of the Markov-blanket property to conduct *group-query* processing for making even more effective query plans. When the Markov blankets of multiple queries overlap, we are provided with more inter-query relations to further reduce the overall resource usage. For instance, Figure 2(a) shows that when two queries arrive over “water temperature” and “air temperature,” respectively, the BN can tell us that we can treat these two queries as one *group query* because of their overlapping Markov blankets. The BN model thus can offer scalable performance to a large number of queries, as well as a large number of sensors.

In addition to supporting traditional *aggregate queries* in a more efficient and effective way, the employment of BN enjoys two additional benefits. First, a new class of *diagnostic queries* can be supported. A BN can readily use the sensor dependencies to determine the cause of abnormality in sensor data. This is because from the sensor-attribute relationships, we know the dependencies between different attributes, which can be used to detect an exception and then determine its type. Let us revisit Figure 2(a). The attribute “air temperature” is dependent on “water temperature.” When readings of the “air temperature” sensor are out of its normal range, the BN tells us that we can query the values of “water temperature” (i.e., the only node in its Markov Blanket) to determine whether the “air temperature” is truly abnormal, or the air-temperature-sensor has malfunctioned. If the readings of “water temperature” are also “abnormal,” but air- and water-temperature readings exhibit high conditional probability, then we can say that the air-temperature-sensor is normal and that the air temperature is abnormal. A traditional correlation-based scheme must verify the correlations with all sensors, and thus does not scale well. Second, the intuitive BN representation can be used to identify “hot spots” and replicate inexpensive and highly acquired sensors to improve both query efficiency and network reliability. For example, a sensor attribute with a high node degree (say “wind speed” in Figure 2(a)) in the BN and with low acquisition cost is likely to be acquired more often as it provides information about many other nodes at a reduced cost. Since such a *hot node* is likely to experience heavy network traffic, it should be replicated in the network.

In summary, this paper makes the following three important contributions:

1. We propose a compact and accurate abstraction of sensor networks based on the BN model. The employment of BN also permits us to tap into the work of sequential BN update, which can adapt BN structure and parameters to newly arrived data.
2. We devise a query plan generation algorithm that can save precious communication and computation resources in answering *group queries* with desired accuracy. Our

experimental results show that our proposed scheme outperforms the correlation-based model for exploiting sensor dependencies (thanks to the Markov-blanket property) to conserve resources.

3. In addition to traditional aggregate queries, our model can answer a new class of *diagnostic queries* for fault detection and data inference, and can also assist sensor deployment and configuration.

The rest of the paper is organized as follows. We discuss related work in Section 2. In Section 3, we describe our sensor network architecture showing how a sensor network in its most general form can be represented as a BN and can be used to address queries. Section 3.2 presents the details of our query-plan-generation algorithm using the Markov-blanket property. We describe adaptations of Bayesian inference mechanisms to address general user queries as well as diagnostic queries. The experimental testbed and validations are described in Section 4. Section 5 presents concluding remarks.

2 Related Work

Our work is most related to the research work proposed in [9][10]. The authors propose to learn a probabilistic model that captures the correlations that might exist between different sensor attributes. The learned model then aids in generating plans for answering queries at a lower cost. Such a correlation model builds one joint distribution table over all the sensor attributes and infers the probabilistic value of an attribute by conditioning all the other attributes on it. This approach does not scale well, as we have pointed out in Section 1. Our BN offers a compact representation, which not only makes query-plan generation efficient and scalable, but also allows *query grouping* to conserve even more resources.

Approximate query answering methods such as approximate caching [23] or DKF [17] can also be used to predict the value of a queried variable within bounded thresholds using the temporal dependencies. However, such schemes continuously monitor the streaming variable (to check if it is within the bounds) making them less effective in terms of energy conservation.

The probabilistic approach of query answering has also been studied in the recent past for moving object databases in [4]. While these efforts can produce effective solutions for providing probabilistic query results (typically object location) over imprecise data, their solutions are not directly applicable to sensor networks that operate in a resource-constrained environment. Furthermore, these solutions are limited to query inference, whereas we also present algorithms to generate query plans using a confidence-driven principle.

Recent works in [20][26] propose sensor-database query models for the declarative, SQL-like query paradigm. The focus of these efforts is also to maximize in-network query processing to reduce sensor resource usage while still meeting the query precision specifications. These query models do not exploit sensor-attribute dependencies and hence cannot handle *group-query-plan generation* or *diagnostic queries* efficiently.

BNs have been used in traditional database systems mainly for attribute-selectivity estimation [13] and data-mining purposes. For sensor networks, BNs have primarily been used for sensor fusion and estimation [21]. Recent work in [2] employs BNs for

probabilistic inference in sensor networks to determine if a sensor is surrounded by *enemy agents*. Our work, to the best of our knowledge, is the first that employs BNs for sensor-query processing.

BNs have found applications in diverse fields like biology, computer vision, computer software and decision support systems to name a few [22]. Since it interests a lot of different communities, BN research has been looked into from the perspectives of structure learning [11][16], parameter estimation [12], efficient inference schemes [15][19][24], batch-mode and online updating of structure and parameters [11][1]. Due to space limitations we cannot have a detailed discussion of all of them however we provide pointers to interested readers.

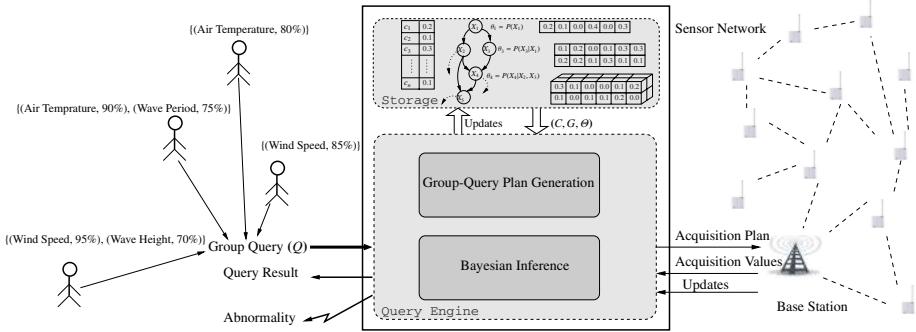


Fig. 3. Sensor Network Architecture

3 Architecture and Model

Figure 3 presents a typical sensor-network architecture, which consists of sensors (on the right-hand side), a query engine (in the middle), and queries issued by multiple users (on the left-hand side). The queries are formulated into a group-query \mathbf{Q} , which we define as follows:

Definition 1. Group Query. A group query \mathbf{Q} , in a sensor network of n attributes can be represented as:

$$\mathbf{Q} = \{(X_i, \delta_i) | (X_i \in \mathbf{X}) \wedge (0 \leq \delta_i \leq 1) \wedge (1 \leq i \leq n)\} \text{ s.t., } \delta_i < \max_l P(X_i = x_{il}) \quad (1)$$

where $\mathbf{X} = \{X_1, X_2, X_3, \dots, X_n\}$ is the set of sensor attributes, δ_i is the confidence requirement for reporting the value of X_i and, $P(X_i = x_{il})$ is the probability with which X_i assumes a value of x_{il} . (A variable X_i in the BN is discretized into k_i bins. The subscript variable l takes values such that $1 \leq l \leq k_i$.)

For group-query \mathbf{Q} , the query engine generates a query plan, and then acquires data from the sensors. To generate a query plan for \mathbf{Q} , the query engine consults the BN, which models the sensor network as a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$. The vertex set \mathbf{V} is a set of random variables $\{X_i\}$ (one for each sensor), and the directional edge set \mathbf{E} captures the statistical relations between sensors. \mathbf{E} is a subset of \mathbf{e}_{X_i, X_j} , where $\mathbf{e}_{X_i, X_j} = \{(X_i, X_j), 1 \leq i, j \leq n, i \neq j\}$, represents the *influence relation* between (X_i, X_j) . The cost set $\mathbf{C} =$

$\{c_1, c_2, c_3, \dots, c_n\}$ holds the acquisition cost associated with each node. The values in the cost set are functions of the routing and sensing costs. Changes to the BN and the cost sets (though rare) are updated when necessary. Based on the BN and data-acquisition costs, the query engine generates a query plan for group query \mathbf{Q} such that the confidence requirement (δ_i) in reporting the values of queried attributes (X_i) can be met while consuming minimum sensor resources.

In the remainder of this section, we first briefly present the BN construction details. We then present our approach to generate efficient query plans for addressing group-queries. Finally, we present how BN can be used to answer traditional aggregate and diagnostic queries. Due to space constraints, we present only the critical steps of our methods. Detailed descriptions can be found in [18].

3.1 Bayesian Network Construction

A BN in its most general form consists of two parts: *model structure* and *model parameters*. The *model structure* is a directed graph in which the vertices are random variables and the directed edges represent the causal relationships between variables. To reduce the learning and inference complexity on BNs [11], we restrict the graph structure to be a polytree, which is a Directed Acyclic Graph (DAG) where an internal node can have any number of parents and children as long as there is no cycle in the corresponding undirected graph. The *model parameters* are mainly the Conditional Probability Tables (CPTs), where each CPT summarizes the conditional probability distribution of a nodal variable given its parents. The CPT of a node without any parent is its prior.

Bayesian inference uses a likelihood definition which states that a good model is one that is likely and can explain data well. Whether a model fits data well can usually be validated objectively based on a cost function. In information theory [6], causality is measured by the reduction in the uncertainty or *entropy* of a random variable, given others. Hence, a reasonable definition of a good fit (a reasonable cost function) for a Bayesian network is the total entropy reduction (or the amount of entropy left) given the set of chosen edges. For any variable X_i , the entropy $H(X_i)$ is defined as follows:

$$H(X_i) = - \sum_{l=1}^{k_i} P(X_i = x_{il}) * \log(P(X_i = x_{il})). \quad (2)$$

Structure learning is mathematically a constrained optimization problem: maximizing entropy reduction subject to the desire of using a simple network. We therefore construct our Bayesian tree minimizing the total entropy in the network ($\sum_{i=1}^n H(X_i|Parents(X_i))$), where $H(X_i|Parents(X_i))$ measures the entropy of X_i given its parents. We first construct a completely connected digraph such that $w_{X_i, X_j} = H(X_i|X_j)$ where w_{X_i, X_j} is the weight of the edge e_{X_i, X_j} and then build a directed minimum spanning tree in $O(n^2 \log(n))$ time using the algorithm described in [5].

After the network structure has been learned, the remaining uncertainty lies only in specifying the CPTs (parameter learning). We model conditional distributions in a CPT using probabilistic distributions in the exponential family (e.g., binomial, multinomial, and beta distributions for discrete random variables, or Gaussian and multivariate-Gaussian distributions for continuous variables) [16]. The advantage of using distributions in the exponential family is that a *closed-form* solution to parameter learning problem can be obtained using either maximum-likelihood or Bayesian learning.

There have been many independent research efforts on algorithm development for learning a BN model (e.g., [11]), and for conducting Bayesian inference (e.g., [7][16][24]). Since the focus of this work is to use the BN to conduct sensor queries, but not to devise new BN-related algorithms, we employ representative algorithms for BN generation [5] and inference.¹

3.2 Query Plan Generation

We now present our approach for generating group-query plans, and our techniques for addressing *diagnostic queries*. One of the most important properties of the BN that forms the theoretical foundation of our proposed algorithms is the conditional independence on the *Markov blanket* [24] (also called Markov Condition), which is defined as follows:

Definition 2. Markov Blanket. Given a BN G , the Markov blanket $MB(X_j)$ of a node X_j in G is the set of nodes that are made up of X_j 's parents, its children and other parents of its children.

Given $MB(X_j)$, X_j is conditionally independent of all other variables in G . This can be mathematically represented as shown in Equation 3.

$$P(X_j|G) = P(X_j|MB(X_j)) = \frac{P(X_j, MB(X_j))}{P(MB(X_j))}. \quad (3)$$

A straight result from Definition 2 is that variables outside the Markov blanket of a variable are not directly relevant for inferring its state once the values of *all* the Market-blanket variables are known. However, the influence of one node on any other node in a BN monotonically decreases (entropy monotonically increases) as the node distance between them increases. This observation is formally stated in the following lemma:

Lemma 1. Given a node X_i and a set of arbitrary nodes \mathbf{Y} in a BN s.t. $MB(X_i) \not\subseteq \{Y \cup X_i\}$, the conditional entropy of X_i given \mathbf{Y} is at least as high as that given its Markov blanket or $H(X_i|\mathbf{Y}) \geq H(X_i|MB(X_i))$.

Proof: We exploit the property of conditional entropy which states that any additional information about a variable cannot cause an increase in its entropy [4], i.e. $H(X_i|X_j, X_k) \leq H(X_i|X_j)$ or $H(X_i|X_k)$. Further, separating $MB(X_i)$ into two parts: $\mathbf{MB}_1 = MB(X_i) \cap \mathbf{Y}$ and $\mathbf{MB}_2 = MB(X_i) - \mathbf{MB}_1$, and denoting $\mathbf{Z} = \mathbf{Y} - MB(X_i)$, we have:

$$\begin{aligned} H(X_i|\mathbf{Y}) &= H(X_i|\mathbf{Z}, \mathbf{MB}_1) && \because \mathbf{Y} = \mathbf{Z} \cup \mathbf{MB}_1 \\ &\geq H(X_i|\mathbf{Z}, \mathbf{MB}_1, \mathbf{MB}_2) && \because \text{Additional information cannot increase entropy} \\ &= H(X_i|\mathbf{Z}, MB(X_i)) && \because MB(X_i) = \mathbf{MB}_1 \cup \mathbf{MB}_2 \\ &= H(X_i|MB(X_i)). && \because \text{Markov-blanket definition} \end{aligned}$$

Equipped with lemma 1 and the conditional independence property of the Markov blanket shown in Equation 3 we can now present our group-query plan generation algorithm. A group-query plan replaces the original queries with queries on alternate sensors that can meet the specified query confidence-levels at a reduced cost. Which sensors to query should depend on at least two factors: how expensive it is to query a particular sensor, and how much information the new sensor data can provide about

¹ We employ Pearl's message passing [24] algorithm for Bayesian inference.

the queried attributes. While the answer to the first question is readily available from the cost set \mathbf{C} , the answer to the second question derives from Lemma 1: For a query attribute X_i in G , its state is influenced directly by every variable in its Markov blanket. Thus, to improve the confidence level of X_i , we analyze attributes from its Markov blanket. Since variables in the Markov blanket can in turn be affected by variables in *their* Markov blankets we might need to analyze Markov blankets recursively. However, since the confidence level typically dies out quickly beyond the immediate Markov blanket of the query object, recursive attribute-analysis rarely takes place in practice.

We consider a confidence-driven query paradigm where a group-query \mathbf{Q} may query several attributes, each with some minimum confidence requirement. The amount of information that sensor \mathbf{X}_i (in the Markov Blanket of a queried sensor) can provide about another sensor $\mathbf{X}_j \in \mathbf{Q}$, is quantitatively available as the conditional entropy reduction i.e. $H(\mathbf{X}_j) - H(\mathbf{X}_j|\mathbf{X}_i)$. Thus, we decide upon which sensors to query, by selecting them in a greedy fashion so to maximize the overall entropy reduction at least possible acquisition cost, such that the confidence requirement of all queried attributes are met. The details of this sensor selection algorithm can be found in [8]. Once the states of the variables (or sensor attributes) to be acquired are available, we can use that information to answer a wide variety of queries. The ability to answer a variety of queries will hinge upon one critical element: the ability to determine the likely state of a BN. As the BN description is essentially a probabilistic one, the state descriptions will also be expressed in terms of a likelihood function conditioned upon prior, current state, and sensor data.

We partition the set of nodes in a BN into three classes: the set of queried attributes (denoted as \mathbf{X}_q), the set of attributes whose values are known (denoted as \mathbf{X}_e), and the rest of the attributes (denoted as \mathbf{X}_h). Both \mathbf{X}_e and \mathbf{X}_h can be empty. In the case that \mathbf{X}_e is empty, we do not have any sensor data, so the inference will rely completely on the prior (or historical data and trend). In the case that \mathbf{X}_h is empty, all sensors are to be queried, and we obtain very detailed knowledge of the network to make inferences. In other cases (which are the typical cases), \mathbf{X}_q is obtained from user query \mathbf{Q} , and \mathbf{X}_e is obtained from Υ . The probability of a queried variable $X_i \in \mathbf{X}_q$ can then be obtained as:

$$P(X_i|\mathbf{X}_e, G) = \frac{P(\mathbf{X}_e|X_i, G)P(X_i|G)}{P(\mathbf{X}_e|G)}, \quad (4)$$

where $P(X_i|G)$ is the prior probability of the queried variable, $P(\mathbf{X}_e|G)$ is the marginal probability of the evidence, and $P(\mathbf{X}_e|X_i, G)$ is likelihood.²

The *pdfs* of the queried attributes obtained in Equation 4 (as a result of Bayesian inference) can then be used to answer a variety value and aggregate queries (refer [8] for details). For example, a range query to compute the probability of \mathbf{X}_i lying in range $[l_i, u_i]$ can be computed as: $P(X_i \in [l_i, u_i]) = \int_{l_i}^{u_i} p(x_i)dx_i$.

3.3 Answering Diagnostic Queries

We perceive data abnormality as an event whose likelihood is suspiciously small, given the historical trends and current network state. For example, in the NDBC datasets,

² A variety of real-time Bayesian inference algorithms are available [15], and the choice of a particular inference algorithm is beyond the scope of this paper.

if the historical trends suggest that “sea temperature” is always 5 – 10% lower than the “air temperature,” then situations would be abnormal when the temperature difference between the two sensors exceeds the threshold. The two main reasons for such abnormality are:

- Failures: A fault in the sensor or communication mechanism causes arbitrary values to be reported at the server, or
- Emergence or dissolution of statistical relations: New attribute dependencies may have evolved affecting the historical likelihood of events.

Suppose the BN topology and the CPTs do not change. A naive method for detecting abnormality is to bound the expected sensor values or attribute correlations, and reporting abnormalities if the observed measures fall out of bounds. This approach does not work well for at least three reasons. First, this method is not scalable in large networks where an attribute is dependent on many other attributes. Second, historical data can reveal that high temperature differences *might* occur. Third, an abnormal reading on one sensor in an extreme weather condition can be an accurate reading, not necessarily resulting from the sensor’s failure or a communication fault. Therefore, abnormality depends on the joint likelihood of several events *under a given BN state*.

We propose a *trigger & verify* approach to detect abnormalities in our sensor network architecture. Each time a value from a sensor is received at the query evaluator, it checks to see whether such a value is likely to be seen under the *current BN state*. This likelihood measure is available at no extra cost from the Bayesian inference engine (shown in Equation 4). If the likelihood measure for any attribute X_i is suspiciously small, the query evaluator *triggers* a request for abnormality diagnosis at each of the nodes in $MB(X_i)$. All nodes receiving a request for abnormality diagnosis capture a continuous sequence of data values and compute the likelihood of observing such a sequence as follows:

Suppose sensor X_j , discretized into bins $\{b_{j1}, b_{j1}, b_{j3}, \dots, b_{jk_j}\}$, is serving an abnormality diagnosis request. It first captures a sequence S_j of continuous observation³ and then obtains the event counts, i.e., the number of times X_j falls in bin b_{ji} and so on. Let the counts be denoted by $\{\alpha_{j1}, \alpha_{j2}, \alpha_{j3}, \dots, \alpha_{jk_j}\}$, then the likelihood of observing sequence S_j given G is

$$L(S_j|G) = \left(\sum_{i=1}^{k_j} \alpha_{ji}! \right) \prod_{i=1}^{k_j} \frac{(p_{ji})^{\alpha_{ji}}}{\alpha_{ji}!} , \quad (5)$$

where, $p_{ji} = P(x_j = b_{ji})$ and $\sum_{i=1}^{k_j} \alpha_{ij} = |S_j|$. If $L(S_j|G)$ is small enough then the event is *verified* as abnormal and is reported back to the server. A direct consequence of such a trigger & verify approach is that 1) a trigger generated due to transient communication breakdown will not be verified as an abnormality, 2) broken sensors (those with a faulty sensing device) will verify abnormalities, and 3) sensors in the Markov blanket of the broken sensor will not verify the abnormality.

If an abnormality is detected in some sensor readings, the next logical questions to ask are “What caused the abnormality?” and “How will the abnormality affect other

³ Here, we make two simplified assumptions on S_j . First, S_j is a sequence of independent, identically distributed random variables forming a multinomial distribution. Second, the state of the network shows little or no variation for the duration of collecting S_j .

Table 1. Relative Costs of Sensors for Real Datasets

NDBC Data							Intel Data					
Attribute	WP	AP	SP	WT	AT	WH	DR	Attribute	T	H	L	V
Rel. Cost (C)	0.14	0.08	0.22	0.27	0.09	0.10	0.10	Rel. Cost (C)	0.328	0.328	0.344	0.001

sensor readings?" While we already addressed the first question, the second question can be addressed using the *model structure* on the BN. When an X_j sensor is detected as broken, we infer its value from its Markov blanket in the BN graph. Furthermore, X_j is no longer used to infer values of variables that contain X_j in their Markov blankets.

4 Experimental Validation

We evaluated the performance of our query engine using BN on three datasets (depicted in Section 4.1). In this paper, our experimental analysis is organized into four parts, we have a more detailed analysis available in [18]. The first experiment examined the effect of group query size and confidence requirements respectively, on resource conservation (Section 4.2). We also compared the resource savings against those obtained using correlation model under varying query-confidence levels (Section 4.2). In the second experiment we analyzed the query answer quality achieved using our proposed approach (Section 4.3). The third experiment analyzed the abnormality detection ability of our proposed model (Section 4.4). The last experiment studied selectivity of attributes under different query conditions (Section 4.5).

4.1 Experiment Setup

We used two real datasets and one synthetically generated, as described below:

- *NDBC Dataset* – This real-world dataset was obtained from the National Data Buoy Center (NDBC) [25]. The sensor network consists of numerous ocean buoys streaming data of different modalities every hour to a base station. The data have seven attributes: “average wave period” (WP), “air pressure at sea level” (AP), “wind speed” (SP), “water temperature” (WT), “air temperature” (AT), “wave height” (WH) and “wind direction” (DR) with relative costs shown in Table I. We used data from three buoys in the San Francisco area (Station IDs 46012, 46013, 46026) in all our experiments. Historical data dating from year 1981 to 2003 were used for learning (with discretization into 4 bins), and segments of year 2004 data were used for testing.
- *Intel Data* – This real dataset (also used in [9][8]) was obtained from the Intel Research, Berkeley Lab [3]. The data were collected using 54 sensors providing “temperature” (T), “humidity” (H), “light” (L) and “voltage” (V) measurements (relative costs are shown in Table I). We used 50% of the data for testing after discretizing each attribute into 8 bins.
- *Synthetic Data* – The synthetic datasets were constructed for the purposes of rigorous testing and for evaluating the performance under ideal conditions. We tested the system on several such datasets; but due to space constraints we report results on one which had the following properties: The BN was generated with 50 nodes having a

maximum node degree of eight. Each node was discretized into five bins, and the CPTs were generated according to the Dirichlet distribution⁴ with $\lambda = 0.01$.

Cost functions associated with different attributes were available for the real-world datasets. For synthetic data we tested the performance for randomly generated cost functions.

Our testbed consisted of 1,500 group queries for NDBC data, 5,000 group queries for Intel data, and 2,500 for synthetic data; all were selected randomly from the testing data such that no two successive queries were separated by more than 10 units of time. The random selection ensures that the results are not biased toward particular temporal query patterns. Once a sensor value is available at the central server, we let its uncertainty value grow exponentially with time, which is taken into account using Bayesian inference.

We first define a few parameters that were used in the experimental setup to evaluate the performance of the system:

- Group-query size ($|Q|$) – The number of sensor attributes whose values are required by one or more users at a time. The maximum value of $|Q|$ is the number of nodes in the BN (e.g. $\max(|Q|) = 7$ for NDBC data).
- Confidence requirement – (δ_{min}) – The confidence required in reporting the values of the attributes in $|Q|$ ⁵.

4.2 Resource Conservation

We define resource conservation as the percentage of the total resources saved in the sensor network to address all queries over the resource consumption if all the queried attributes in Q were to be acquired directly.

Effect of Grouping Queries. In Figure 4, we compare the resource conservation achieved in addressing group queries using our group-query plan algorithm (the upper plane with solid lines) as opposed to processing them individually (the lower plane with dotted lines). We show the results for both real datasets as we vary both $|Q|$ (from 1 to $\max(|Q|)$) and δ_{min} (from 0.80 to 0.98). In the figure, the vertical axis shows the resource conservation; the lighter the shaded color on the plane, the higher the resource conservation achieved. Note that even when issuing queries individually (group-query size $|Q| = 1$, along the left-edge of the figure), using BN inferencing can already save

⁴ If p_i is the probability of choosing a collection of items of size i from an item-set of size n , then $\{p_1, p_2, \dots, p_n\}$ are the parameters of the multinomial distribution. The Dirichlet distribution with parameter λ is the conjugate prior on the parameters of the multinomial distribution. $\lambda \ll 1$ encourages “deterministic” CPTs (one entry near 1, the rest near 0), $\lambda = 1$ causes entries to be drawn from $U[0, 1]$ and $\lambda \gg 1$ causes all entries to near $1/k$ where k , is the discretization size [12].

⁵ To maintain uniformity we always choose to query $|Q|$ costliest attributes when the group-query size is less than its maximum value. We query all attributes in Q with the same value of δ_{min} . For example, for $|Q| = 2$ in NDBC data, we always choose WT and SP and query both of them with 90% confidence for $\delta_{min} = 0.90$.

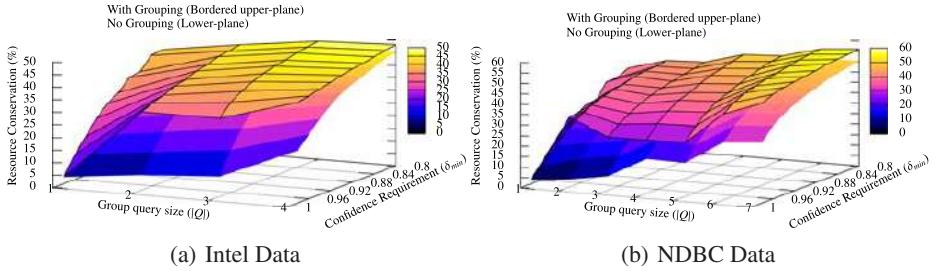


Fig. 4. Resource conservation as a function of δ_{min} and $|Q|$

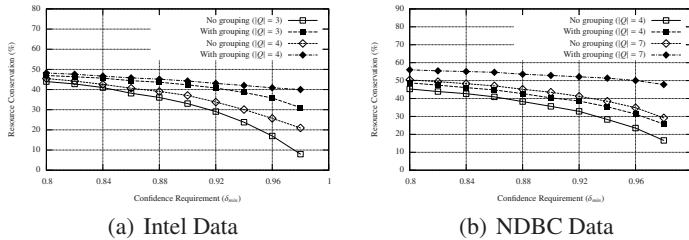


Fig. 5. Resource conservation as a function of δ_{min}

significant resources. (We will present *2D* figures shortly to highlight some results.) When queries were grouped using our group-query algorithm, the resource conservation was more significant.

To facilitate a better view, Figure 5 provides a 2D view on the resource conservation with different group sizes. Figure 5(a) shows that when $|Q| = 4$ for the Intel dataset, the savings at various confidence levels are consistently achieve above 30%. This is because a larger group size provides the algorithm with more room to use the inter-attribute dependencies more effectively. On the NDBC data, Figure 5(b) shows that the savings can be above 50% when $|Q| = 7$ and $\delta_{min} \leq 0.96$.

BN vs. Correlation Model. An important question to answer is “how does resource conservation of using BN compare with not using BN (or using the traditional correlation model)?” It is evident that using the correlation model incurs higher computational cost due to the model’s larger search space for alternate sources. We were curious to find out whether the employment of BN could improve resource conservation; and if so, what factors contributed to the improvement.

We compared the resource conservation obtained for group-queries using our proposed approach against the one that uses a joint probability distribution over all the variables. This is similar to one proposed in [8] (which is equivalent to having completely connected graph structures as shown in Figure 1). Figure 6 shows the percentage of resource conservation on the two real-life datasets for $\delta_{min} = 0.90$. For Intel data, at $|Q| = 1$ (i.e. when there is no possibility of grouping queries), using BN conserves about 3% more resources than the correlation model. When we increase the group size,

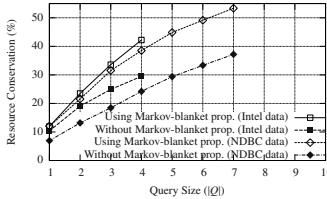


Fig. 6. Resource conservation with $\delta_{\min} = 0.90$

the conservation increases (at $|Q| = 4$, using BN achieves 12 additional percentile of conservation). A similar pattern is obtained from the NDBC dataset. The additional conservation achieved by BN is due to the fact that two correlated attributes might be conditionally independent on a *cheaper* attribute, a property that the Markov blanket decodes successfully, but the correlation-based model fails to decode. (Setting δ_{\min} at different levels achieves the same result: BN outperforms the correlation model in resource conservation.)

4.3 Query Answer Quality-Loss

Our BN framework provides query answers based on probabilistic estimates. Therefore, as with the correlation model proposed in [8], 100% query precision cannot be attained. An effective query plan is one that conserves resources and meets the confidence values of the queries most of the time. We define *loss* as an event when the confidence of a queried attribute has not met the requirement after the execution of the query plan. For a particular Q , we calculated the *quality-loss* as the per-attribute mean loss percentage (or the percentage of the total losses over all attributes to the total number of times they were queried). Quality-loss depends on how tightly the data distribution fits the one used in the BN. If the data dependencies are modeled accurately in the CPTs, the quality-loss should not be affected adversely by $|Q|$ and δ_{\min} .

We show the quality-loss observed for different datasets in Figure 7 for $\delta_{\min} = .90$. All other testing parameters were the same as described for the earlier experiments. Since real-data distribution can be modeled only to a certain degree of accuracy, we observe slight variations in quality-loss as we increase $|Q|$. However, as seen in Figure 7(a), the loss always stays well under 7% in all cases. For the synthetic dataset (Figure 7(b)) we observe much less variation (always less than 1%), because the data distribution was modeled accurately. As discussed earlier, large $|Q|$ allows our group-query algorithm to exploit sensor dependencies more effectively for producing stronger probabilistic estimates, and hence achieving lower quality-loss.

There is a trade-off between the quality-loss and the confidence requirement. High-confidence queries would cause the quality-loss to rise, though graceful degradation is desired. In another experiment, we studied the effect on quality-loss of increasing δ_{\min} . The results obtained for all three datasets as we increased the confidence requirement from 0.80 to 0.98 are shown in Figure 7(c). The slow degradation shows the effectiveness of our approach, with a loss of less than 8% even at 98% confidence requirement for real datasets. The quality-loss for synthetic data always remains under 1%.

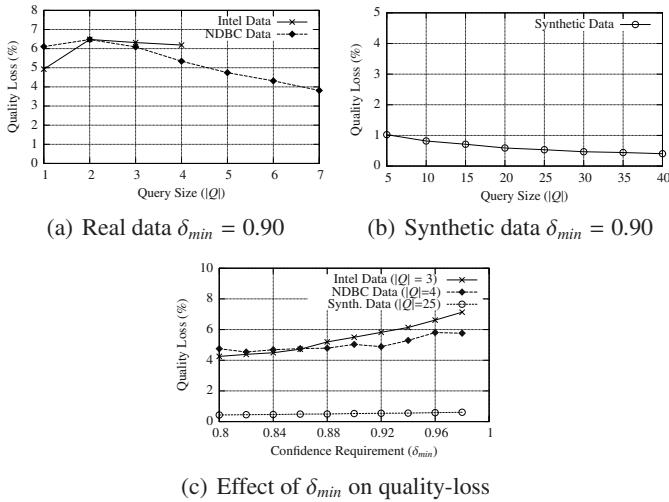
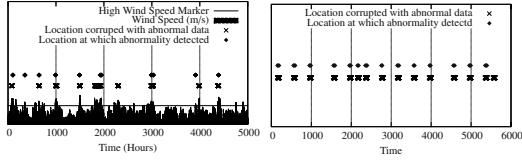


Fig. 7. Quality loss

4.4 Abnormality

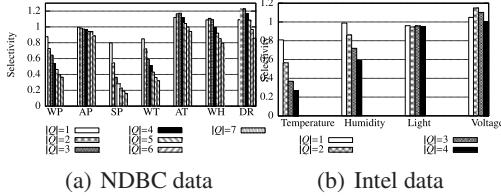
We tested the abnormality-detection ability of our model as proposed in Section 3.3. This abnormality experiment required domain knowledge on the data. We tested our proposed approach on the NDBC real-world dataset as follows: The NDBC facilitates the search for extreme weather conditions over its entire historical database. Since these extreme weather conditions are so classified by domain experts, we can safely tag them as “abnormal events.” We searched for extreme “high wind” conditions over the entire historical dataset in San Francisco County and corrupted a normal testing dataset (used in the experiments described earlier) with 12 extreme conditions at randomly selected locations. We then modified the query-plan generation algorithm such that the “wind speed” attribute was always selected to be acquired by the query plan. We validated our approach by observing if (1) our algorithm could detect all the abnormal events, and (2) if it would correctly detect the time at which the abnormality occurred. Figure 8(a) shows the abnormality detection behavior of our algorithm with $|S_j| = 10$. The horizontal line shows the threshold for the discretization used for “wind speed” such that all values above it would fall in the same discretization bin. As seen in the figure, our model catches all the abnormal events except for one (the sixth from left) and reports one *normal* event (the second from left) as an abnormality.

Though there are many locations in the graph where the wind speed exceeds the threshold, our algorithm detects only those that were corrupted manually. Thus, our model is quite effective in detecting abnormalities and in reducing false positives. Abnormality-detection results for the synthetic dataset are shown in Figure 8. We generated a BN similar to the one used in testing but with a different Dirichlet distribution ($\lambda = 1$) for the CPTs. We corrupted the normal testing data with data sampled from the new BN for a randomly selected attribute at 15 different locations. Verification sequence was set to $|S_j| = 10$. As seen in the figure, our algorithm captures all the abnormalities except for one (the last one).



(a) NDBC data

(b) Synthetic data

Fig. 8. Abnormality Detection**Fig. 9.** Selectivity with $\delta_{\min} = 0.90$

4.5 Selectivity

The selectivity of an attribute is the ratio of the number of times it is acquired to the total number of times it appeared in some query. The selectivity pattern can be extremely useful in improving network reliability and identifying “hot-spots” as discussed in Section II. Sensors with high selectivity attributes should be replicated more in the network, and the communication overlay network could be adjusted so that energy consumption is reduced. A cheap attribute, with a high node degree in the BN graph is likely to show high selectivity since it lies in the Markov blanket of many other nodes and thus provides information about other nodes at a low cost. On the other hand, a costly attribute (with a high node degree) is likely to show low selectivity if there are other less inexpensive nodes lying in its Markov blanket. We show the selectivity behavior of the two real-world datasets in Figure 9. The experiment parameters were the same as those used in the resource conservation experiments. We expect to reduce the selectivity as we increase $|Q|$, since as it allows scope for better optimization. As seen in Figure 9(b) the selectivity of “temperature” and “humidity”(having node degree 2) drop significantly as compared to “voltage”. This is due to the fact the relative acquisition cost of “voltage” was very low, making its acquisition more frequent than costly attributes. Selectivity for “light” does not drop because it does not have any low cost attribute in its Markov blanket. Figure 9(a) shows the selectivity graph for the NDBC dataset. We observe that the selectivity of “speed” and “wave period” (costly attributes) drops significantly with the increase in $|Q|$, as they have high node degrees. The node degree of “speed”, being the highest (Figure 2(a)), shows the sharpest fall.

5 Conclusions and Future Work

In this paper, we have proposed using BNs for characterizing sensor networks for probabilistic query answering and sensor diagnosis. We proposed a greedy algorithm for

saving sensor resources by grouping individual queries into one group-query. Our approach uses a Bayesian inferencing scheme which, in addition to providing probabilistic estimates of the queried variables, also provides effective methods for the sensor network diagnosis. The *pdf*'s of the queried variable can be used to address a wide range of value and aggregate queries. The BN structure also helps in improving the sensor network infrastructure by providing an intuitive model of the inter-attribute dependencies. Through examples and experiments on both real and synthetic datasets, we demonstrated that the BN is more effective in saving sensor resources than the previously proposed simplistic probabilistic models using correlations. Our model provides significant improvement in resource conservation of 15 – 20% over traditional models. We also showed the effectiveness of our model in capturing abnormalities and predicting attribute selectivity.

We plan to extend our work to address *what-if* queries. Bayesian inference allows us to reason about hypothetical scenarios given some counterfactual evidence. Such queries are called *what-if* queries. Such queries can be extremely useful predicting state of the network in hypothetical conditions to trigger alarms and issue warnings.

References

1. E. Bauer, D. Koller, and Y. Singer. Update rules for parameter estimation in Bayesian networks. In *UAI'97: Proc. of the Thirteenth Conf. on Uncertainty in Artificial Intelligence*, pages 3–13, August 1997.
2. R. Biswas, S. Thrun, and L. J. Guibas. A probabilistic approach to inference with limited information in sensor networks. In *IPSN'04: Proc. of the 3rd Intl. Symposium on Information Processing in Sensor Networks*.
3. P. Bodik, W. Hong, C. Guestrin, S. Madden, M. Paskin, and R. Thibaux. Intel lab data, <http://db.lcs.mit.edu/labdata/labdata.html>.
4. R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD'03: Proc. of the 2003 ACM SIGMOD Intl. Conf. on Management of data*.
5. C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.
6. T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley and Sons, Inc., 1996.
7. P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-Hard. *Artificial Intelligence*, 60(1):141–153, 1993.
8. A. Deshpande, C. Guestrin, and S. Madden. Using probabilistic models for data management in acquisitional environments. In *CIDR'05: Proc. of 2nd Biennial Conf. on Innovative Data Systems Research*, Asilomar, CA, USA, January 4–7 2005.
9. A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB'04: Proc. of 30th Intl. Conf. on Very Large Data Bases*, Toronto, Canada, September 2004.
10. A. Deshpande, C. Guestrin, S. Madden, and W. Hong. Exploiting correlated attributes in acquisitional query processing. In *ICDE'05: Proc. of 21st Intl. Conf. on Data Engineering*, Tokyo, Japan, April 5–8 2005.
11. N. Friedman and D. Koller. *Tutorial: Learning Bayesian networks from data*. NIPS'01: Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 2001.

12. A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian data analysis*. Chapman & Hall CRC, 1995.
13. L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *SIGMOD'01: Proc. of the 2001 ACM SIGMOD Intl. Conf. on Management of data*, pages 461–472, New York, NY, USA, 2001.
14. R. M. Gray. *Entropy and Information Theory*. Springer-Verlag, New York, 1990.
15. H. Guo and W. H. Hsu. A survey of algorithms for real-time Bayesian network inference. In *AAAI/KDD/UAI'02: Joint Workshop on Real-Time Decision Support and Diagnosis Systems*, Edmonton, Alberta, Canada, July 29 2002.
16. D. Heckerman. Tutorial on learning with Bayesian networks. Technical report, Microsoft Research, March 1995.
17. A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive stream resource management using Kalman filters. In *SIGMOD'04: Proc. of the 2004 ACM SIGMOD Intl. Conf. on Management of data*, pages 11–22, New York, NY, USA, 2004.
18. A. Jain, E. Y. Chang, and Y.-F. Wang. Efficient group and diagnostic queries on sensor networks (<http://www.cs.ucsb.edu/~ankurj/bayesTR.pdf>). Technical report, Computer Science, UC Santa Barbara, August 2006.
19. M. I. Jordan, Z. Ghahramani, T. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.
20. S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: An acquisitional query processing system for sensor networks. *ACM Transactions Database Systems*, 30(1):122–173, 2005.
21. J. Moura, L. Jin, and M. Kleiner. Intelligent sensor fusion: A graphical model approach. In *ICASSP'03: Intl. Conf. on Acoustics, Speech, and Signal Processing*, volume 6, pages 6–10, Hong Kong, April 2003.
22. K. P. Murphy. *Dynamic Bayesian Networks: Representation, inference and learning*. PhD thesis, University of California, Berkeley, Fall 2002.
23. C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, San Diego, California, USA, June 2003.
24. J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.
25. N. W. Service. National Data Buoy Center, <http://www.ndbc.noaa.gov/>.
26. Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR'03: First Biennial Conf. on Innovative Data Systems Research*, Asilomar, CA, January 2003.

Telescope: Zooming to Interesting Skylines

Jongwuk Lee, Gae-won You, and Seung-won Hwang

POSTECH, Pohang, Korea

{julee, gyou, swhwang}@postech.ac.kr

Abstract. As data of an unprecedented scale are becoming accessible, skyline queries have been actively studied lately, to retrieve “interesting” data objects that are not dominated by any other objects, *i.e.*, *skyline objects*. When the dataset is high-dimensional, however, such skyline objects are often too numerous to identify truly interesting objects. This paper studies the “curse of dimensionality” problem in skyline queries. That is, our work complements existing research efforts to address this “curse of dimensionality”, by ranking skyline objects based on *user-specific* qualitative preference. In particular, Algorithm *Telescope* abstracts skyline ranking as a dynamic search over skyline subspaces guided by user-specific preference with correctness and optimality guarantees. Our extensive evaluation results validate the effectiveness and efficiency of Algorithm *Telescope* on both real-life and synthetic data.

1 Introduction

As data of an unprecedented scale are becoming accessible, skyline queries have been actively studied lately, to retrieve “interesting” data objects that are not “dominated” by any other objects, *i.e.*, *skyline objects*. To illustrate, Example 1 shows how skyline queries can be used to identify interesting objects.

Example 1 (Skylines). Consider a basketball coach trying to recruit ideal players excelling in six dimensions d_1, \dots, d_6 , such as the number of games played, rebounds, assists, blocks, steals, and points as illustrated in a toy dataset Table 1. Intuitively, player A is a better choice than B , if A is superior to B in all dimensions, *i.e.*, A dominates B . This coach will thus be recruiting players that are not dominated by any other player, or, skyline objects. In Table 1, all data objects *e.g.*, $\{t_1, t_2, t_3, t_4, t_5\}$ tie as skyline objects, while such objects have different trade-offs. For instance, t_3 is superior to t_1 in 5 dimensions except one, *i.e.*, d_6 . In other words, skyline queries generate numerous ties as they do not judge trade-offs among skyline objects.

As Example 1 illustrates, skyline queries do not require users to specify how to judge trade-offs across dimensions. While such property makes it simpler for users to formulate a query, at the same time, by not differentiating skyline objects with varying trade-offs, *i.e.*, treating all as *ties*, the number of skylines is often too numerous to identify truly promising objects with respect to user preference, especially, when the number of dimensions is large, *i.e.*, “curse of

Table 1. Toy dataset for Example 1

	d_1	d_2	d_3	d_4	d_5	d_6
t_1	5	2	1	1	2	5
t_2	3	1	2	2	5	4
t_3	7	3	5	3	4	1
t_4	1	4	4	4	1	3
t_5	2	5	3	6	3	2

dimensionality”. It is thus non-trivial to identify truly interesting objects from many skylines.

Recently, there have been research efforts to address this “curse of dimensionality” problem, focusing on a specific subset of the skylines, in the following two directions: First, references [12] propose to find more interesting skylines by identifying skylines for a subspace of total dimensions. For instance, in Table 1, if we compute the skylines of some subspace, say, $\{d_1, d_2, d_3\}$, some tuples are no longer skylines, *e.g.*, t_2 is dominated by t_3 . Based on this observation to identify interesting subsets of skylines, references [12] study computation of skylines in varying subspaces—More specifically, reference [1] studies how to amortize the cost of computing skylines over all possible subspaces, while reference [2] studies how to identify *decisive* subspaces for each skyline object. However, these works do not study how to use subspace for generating a result with desirable size. References [34] thus study how to rank skyline objects using various ranking criteria based on subspace skylines. However, the proposed ranking criteria cannot adapt to user-specific information needs, *i.e.*, *user-oblivious*.

This paper combines the advantages of the two approaches. First, our framework alleviates users from identifying an appropriate subspace, which we automatically identify for user-specified preference and retrieval size. Second, unlike user-oblivious ranking approach, we adapt to user-specific information needs. Further, the information we obtain from users is highly intuitive, *i.e.*, a qualitative partial ranking over data dimensions. For instance, in Example 1, a coach trying to recruit defense players can simply specify he takes rebounds more seriously than game points, *i.e.*, *rebound > game points*. Summing up, our work complements existing skyline works by ranking skylines based on user-specific criteria and retrieval size. In a clear contrast, our work distinguishes itself from rank query processing [56789] as well, which requires users to specify a complete and quantitative ranking function for all tuples, which is often much harder to formulate than qualitative partial preference used in our framework, as pointed out in [10]. In particular, our proposed Algorithm *Telescope* abstracts skyline ranking problem as a dynamic search problem over skyline subspaces, guided by user-specified qualitative preference. In summary, we believe this paper has the following contributions:

- We propose the framework to rank skyline objects based on user-specific qualitative preference over dimensions to effectively identify truly interesting points.
- We abstract skyline ranking as a dynamic search over skyline subspaces and develop a dynamic search algorithm guided by user-specified preference and retrieval size. Algorithm *Telescope* is provably correct and optimal.
- We implement Algorithm *Telescope* and extensively evaluate the effectiveness and efficiency over both a real-life dataset and synthetic datasets.

This paper is organized as follows. Section 2 briefly reviews existing efforts related to our work. Section 3 discusses preliminaries on ranking skyline objects based on qualitative user preference over dimensions. In particular, Section 4 proposes Algorithm *Telescope* for ranking skyline. Finally, Section 5 validates the effectiveness and efficiency of Algorithm *Telescope*.

2 Related Work

In many applications, skyline queries have been actively studied to efficiently identify promising objects over the dataset having multiple dimensions. Reference [1] was a pioneering work, which devised block nested loop (*BNL*), divide-and-conquer (*D&C*) and B-tree-based algorithms for identifying skyline objects.

Later, reference [2] followed to improve *D&C* algorithm by pruning dominated objects by partitioning the data space using the nearest neighbor object. Similarly, reference [3] developed sort-filter-skyline (*SFS*) algorithm using pre-sorting lists, which improves existing *BNL* algorithm. However, these algorithms focused on efficient computation of skyline objects and did not study the curse of dimensionality problem.

Recently, there have been research efforts to identify interesting subsets of skylines to address this curse of dimensionality problem, in the following directions: First, reference [4] proposes *skycube* structure, adopting the cube structure of OLAP environments, which amortizes the cost of computing skylines over all possible subspaces. Meanwhile, reference [2] studies how to identify *decisive* subspace for each tuple. However, these works do not study how to identify a result with desirable size. Second, references [3,4] study ranking skyline objects by adopting varying interesting metrics. More specifically, reference [3] uses *skyline frequency* metrics that counts the number of subspaces where each tuple belongs to skylines. Further, reference [4] uses *k-dominate skyline* metrics which considers skylines in *k*-dimensional subspace as well. While these rankings help users to identify more interesting skylines by focusing on the top results in the ranking, the ranking criteria are user-oblivious, which can fail to adapt to user-specific needs.

In summary, this paper complements existing works by identifying more interesting subset of skyline objects with user-specified preference and retrieval size. In particular, Algorithm *Telescope* abstracts the problem as a dynamic search

guided by user-specified preference over numerous skylines, which finds truly interesting skylines with correctness and optimality guarantees.

3 Preliminaries

As illustrated in Example 1, a downside of skyline queries is that the size of skyline objects can be large when dataset is high dimensional. While this “curse of dimensionality” problem has been actively studied lately, existing works do not address the challenge of adapting to user preference in identifying truly interesting objects among the skyline objects. To address this challenge, we define the notion of user preference, and its properties.

Table 2. Notations used

Notation	Definition
\mathcal{S}	The dataset
\mathcal{D}	The dimension set
t_i	A tuple in \mathcal{S}
n	The number of dimensions in \mathcal{S}
d_i	A data dimension ($1 \leq i \leq n$)
\mathcal{W}	User’s preference dimensions ($\mathcal{W} \subseteq \mathcal{D}$)
\mathcal{V}	a subset of preference dimensions \mathcal{W} ($\mathcal{V} \subseteq \mathcal{W}$)
m	The number of user preference dimensions in \mathcal{S}
w_i	A user preference dimension ($1 \leq i \leq m$)
$t_i(d_j)$	The value of a tuple t_i on d_j
$SKY(\mathcal{D})$	Skyline on the dimension space \mathcal{D}

We first formally define *dominate* and *skyline* in Definition 1 and Definition 2, respectively, by using notations introduced in Table 2. These definitions are consistent with the definition of skyline queries used in existing works [1, 2, 3, 4, 11, 12, 13].

Definition 1 (Dominate). A tuple t_i dominates another tuple t_j on \mathcal{D} if and only if $\forall d_k \in \mathcal{D}$, $t_i(d_k) \geq t_j(d_k)$ and $\exists d_s \in \mathcal{D}$, $t_i(d_s) > t_j(d_s)$ ¹.

Definition 2 (Skyline). A tuple t_i is a skyline object on \mathcal{D} if and only if any other tuples $\forall t_j (\neq t_i) \in \mathcal{S}$ do not dominate t_i on \mathcal{D} . $SKY(\mathcal{D})$ denotes the set of skyline objects on \mathcal{D} in \mathcal{S} .

¹ Note that this definition is based on *max* skyline operator [11], while it can be straightforwardly extended to another operator as well, *i.e.*, *min*, where t_i dominates t_j on \mathcal{D} and only if $\forall d_k \in \mathcal{D}$, $t_i(d_k) \leq t_j(d_k)$ and $\exists d_s \in \mathcal{D}$, $t_i(d_s) < t_j(d_s)$.

We then define user preference to be specified by each user. We view user preference as a qualitative ranking of some data dimensions, *i.e.*, *strict partial order* on \mathcal{D} . More formally, we define user preference \mathcal{W} as the *ordered set* of some data dimensions, *i.e.*, $\mathcal{W} = \{w_1, w_2, \dots, w_m\}$ such that $w_1 > w_2 > \dots > w_m$ when w_i is some $d_j \in \mathcal{D}$. We define its semantics as follow: (1) For dimension d_i included in \mathcal{W} , d_i is preferred over any other dimensions not included in \mathcal{W} . (2) Among the dimensions included in \mathcal{W} , preference follows the order of \mathcal{W} . That is, over subspaces having dimensions of same size, the preference is determined *lexicographically*, *e.g.*, $\{w_1, w_2\} > \{w_1, w_3\}$.

Example 2 illustrates this notion intuitively, followed by formal definition of *subset precedence* in Definition 3. Further, based on this subset precedence definition, we rank skyline objects by the precedence of subspace as Theorem 1.

Example 2 (User Preference). Continuing from Example 1 to recruit good guard candidates, the coach considers assists, steals, and game points dimensions are more importantly than the rest of dimensions. Especially, when the coach has preference of assists > steals > points, players with strength in assists and steals will be preferred over those with strength in steals and game points.

Definition 3 (Subset Precedence). For any subset $\mathcal{V} = \{v_1, v_2, \dots, v_{m'}\}$ of \mathcal{W} , *i.e.*, $\mathcal{V} \subseteq \mathcal{W}$, we define a subspace $\mathcal{V}^i = \mathcal{V} - \{v_{m'-i}\}$ of size $m' - 1$ has higher precedence over any other subspaces $\mathcal{V}^j = \mathcal{V} - \{v_{m'-j}\}$ of the same size, if and only if $i < j$, *i.e.*, $v_{m'-i} < v_{m'-j}$.

Theorem 1 (Skyline Preference). For user preference $\mathcal{V} \subseteq \mathcal{W}$, skyline preference follows the order of subset precedence, *i.e.*, $\text{SKY}(\mathcal{V}^0) > \text{SKY}(\mathcal{V}^1) > \dots > \text{SKY}(\mathcal{V}^{m'-1})$ where $\mathcal{V}^i = \mathcal{V} - \{v_{m'-i}\}$. Skyline preference implies (a) already seen skylines outrank the rest to be accessed among unseen skylines, (b) when already seen skylines show again, the rank of skylines follows the subset precedence initially accessed.

With the notion of preference defined, we make observations on key properties of skyline subspaces. These properties observed play essential roles in showing the correctness and optimality of our framework which essentially implies a dynamic search over the lattice structure of subspaces of \mathcal{W} as illustrated in Fig. 1(left). For simplicity, we first assume that every tuple has a different value for each dimension just for now, which are formally defined as *distinct value assumption*. (We will later relax this assumption.)

Definition 4 (Distinct Value Assumption). Any two data t_i, t_j in \mathcal{S} are $\forall d_k \in \mathcal{D}, t_i(d_k) \neq t_j(d_k)$.

With distinct value assumption, previous works have observed that skylines on \mathcal{D} subsume the skylines of its subspace, *i.e.*, *skyline monotonicity* holds, as formally described below.

Theorem 2 (Skyline Monotonicity with Distinct Value Assumption). Given dataset with distinct value assumption, a tuple $t_i \in \text{SKY}(\mathcal{V}')$ is included in $\text{SKY}(\mathcal{V})$ where $\mathcal{V}' \subseteq \mathcal{V} \subseteq \mathcal{D}$.

Proof. See [1].

This monotonicity will be later used to ensure that our framework of exploring skylines subspace lattice (as illustrated in Fig. 1) is correct, *i.e.*, does not return a non-skyline object. However, observe that this monotonicity is conditional to the distinct value assumption, as the example below illustrates.

Example 3 (Example without distinct value assumption). Consider a dataset of 5 objects over 2-dimensions (X, Y) , *i.e.*, $\mathcal{S} = \{a(1, 6), b(3, 6), c(4, 5), d(6, 4), e(6, 2)\}$, which does not satisfy the distinct value assumption. Observe that, while overall skylines on two dimensions $\{X, Y\}$ are $\{b, c, d\}$, skyline on its subspace $\{X\}$ and $\{Y\}$ is $\{d, e\}$ and $\{a, b\}$, respectively. The monotonicity no longer holds as the skylines for $\{X\}$ and $\{Y\}$ are not subsumed by those of $\{X, Y\}$.

While this may seem to compromise the correctness of our framework, we can easily extend the correctness guarantee for any dataset with and without distinct value assumption by replacing the skyline of any subspace \mathcal{V}^i in lattice of Fig. 1 as the intersection with its parent in the lattice, *i.e.*, $SKY(\mathcal{V}^i) \cap SKY(\mathcal{V})$ for its parent \mathcal{V} , as we further discuss in the next section. With this extension, the monotonicity is preserved, as $(SKY(\mathcal{V}^i) \cap SKY(\mathcal{V})) \subseteq SKY(\mathcal{V})$ trivially holds.

4 Algorithm *Telescope*

In this section, we propose our algorithm that identifies truly interesting skyline objects by adapting to user-specific preference and retrieval size. We name our algorithm *Telescope*, for a telescope helping each user to effectively and efficiently focus on interesting skylines depending on user-specific preference. Toward this goal, Algorithm *Telescope* leverages the preliminaries as discussed in Section 3, to rank skyline objects, which is essentially a dynamic search of $2^m - 1$ subspaces considering m preference dimensions among n dimensions.

To illustrate Algorithm *Telescope*, we use the scenario in Example 2. Suppose that a user specifies his/her preference as $\mathcal{W} = \{w_1 = d_3, w_2 = d_5, w_3 = d_6\}$ and retrieval size $k = 3$ for our toy dataset Table 1. As discussed in Section 3, all subspaces of \mathcal{W} can be represented as the lattice graph in Fig. 1(left). We illustrate how Algorithm *Telescope* works in Fig. 1(left): First, we compare the number of skylines of \mathcal{W} , *i.e.*, $|SKY(\mathcal{W})|$, with retrieval size k . Second, we consider the subspace having the highest precedence, *i.e.*, $\mathcal{V}^0 = \{w_1, w_2\}$ and insert its skylines, *i.e.*, $SKY(\mathcal{V}^0) = \{t_2, t_3\}$, into desirable results, and move on to its right sibling, *i.e.*, subspace $\mathcal{V}^1 = \{w_1, w_3\}$, by the order of subset precedence. Third, the number of skylines in \mathcal{V}^0 and \mathcal{V}^1 exceeds the retrieval size k *i.e.*, $|SKY(\mathcal{V}^0) \cup SKY(\mathcal{V}^1)| > k$. Fourth, we zoom into the subspaces of \mathcal{V}^1 , or $\{w_1\}$ and $\{w_3\}$, to identify more desirable k results among them, *e.g.*, $\{t_1, t_2, t_3\}$ by adding $\{t_1\}$ in $\{w_3\}$.

Further, we can transform this lattice graph into a *left-skewed graph* by pruning multiple links to common descendants, as we will formally state in Definition 5. To illustrate, consider adjacent subspaces, *e.g.*, $\{w_1, w_2\}$ and $\{w_1, w_3\}$, sharing

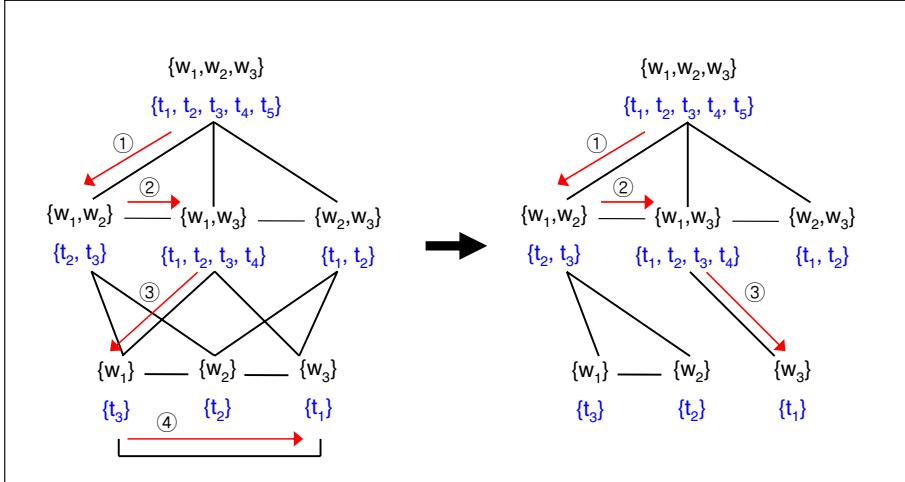


Fig. 1. The lattice graph and left-skewed graph

a common descendant, *e.g.*, $\{w_1\} \subseteq \{w_1, w_2\} \cap \{w_1, w_3\}$. As there are multiple links to $\{w_1\}$ from both $\{w_1, w_2\}$ and $\{w_1, w_3\}$, we keep only the link from the ancestor of higher precedence, *i.e.*, $\{w_1, w_2\}$. With such transformations for every adjacent nodes, we can eliminate multiple links to common descendants by keeping only the link from the highest precedence parent, as Fig. ②(right) illustrates. With this graph transformation, Algorithm *Telescope* guarantees to visit nodes in the descending order of precedence (which will be later used for correctness proof), and at the same time, guarantees not to visit any node twice (which will be later used for optimality proof).

Definition 5 (Graph transformation). *The lattice graph can be transformed into a left-skewed graph, by eliminating a link from adjacent nodes \mathcal{V}^i and \mathcal{V}^j to its common descendant \mathcal{V}^k by keeping only a single link to the common descendant, in particular, the one from the node with the highest precedence.*

By leveraging the left-skewed graph, Fig. ② shows the pseudo-code of Algorithm *Telescope*. We describe how Algorithm *Telescope* works for the user-specified preference \mathcal{W} and retrieval size k as followed:

1. Compare the number of skylines for the root node, *i.e.*, $|\text{SKY}(\mathcal{W})|$, with the retrieval size k .
2. When $|\text{SKY}(\mathcal{W})| > k$, push all subspaces \mathcal{V} of \mathcal{W} with $m - 1$ dimensions in \mathcal{W} into the stack (except for already seen common descendants). The precedence of subspaces will be preserved if pushed from right to left in Fig. ②(right), *i.e.*, low precedence to high, as pushing and popping from the stack will reverse the order.
3. Pop a subspace \mathcal{V} from the stack, and decide whether to insert its skylines $\text{SKY}(\mathcal{V})$ into the results \mathcal{Z} as follow:

- If $|\mathcal{Z} \cup SKY(\mathcal{V})| > k$, go to step 2 to insert its subspaces to the stack, *i.e.*, move on the leftmost child of current node.
- If $|\mathcal{Z} \cup SKY(\mathcal{V})| \leq k$, insert new skyline points $SKY(\mathcal{V})$ into \mathcal{Z} . If $|\mathcal{Z}| < k$ still holds, go to step 3, *i.e.*, move on the right sibling of current node. Otherwise terminate.

Note that, for all child nodes, the monotonicity of skyline (Theorem 2) assures that every data object in the lattice is a part of skylines of \mathcal{W} . Further, when accessing siblings in the order of subset precedence, skyline preference theorem (Theorem 1) ensures the already seen objects outrank the rest to be accessed among unseen objects, *i.e.*, the rank of skylines is decided by the subset precedence initially accessed. Putting together, any search over lattice with two modes of access– (a) to child node with the highest precedence, *i.e.*, leftmost child node, and (b) to sibling node in the decreasing order of precedence, is correct, as we formally state below.

Algorithm *Telescope*($\mathcal{S}, \mathcal{W}, k$)

Input

- \mathcal{S} : dataset
- $\mathcal{W} : \{w_1, w_2, \dots, w_m\}$
- k : retrieval size

Output

- \mathcal{Z} : skylines with respect to k

Procedure

- $\mathcal{T}, \mathcal{U}, \mathcal{V} // Stack, superset of current set, and current set$
- $\mathcal{T} \leftarrow \{\}, \mathcal{Z} \leftarrow \{\} // Initialize the stack and results.$
- **if** $|SKY(\mathcal{W})| > k$ **then**
 - $\mathcal{T}.push(\mathcal{W}) // Push \mathcal{W} into the stack.$
- **while** (\mathcal{T} is not empty **and** $|\mathcal{Z}| < k$)
 - $\mathcal{V} \leftarrow \mathcal{T}.pop() // Traverse subsets of \mathcal{V} if exceed the retrieval size k.$
 - **if** $|\mathcal{Z} \cup SKY(\mathcal{V})| > k$ **then**
 - $\mathcal{U} \leftarrow \mathcal{V} // Keep track of the superset of the current set.$
 - // Insert subsets into the stack except for shared subsets.
 - **for** $i := 0$ **to** $m' - 1$:
 - $\mathcal{T}.push(\mathcal{V}^i) // Push subsets \mathcal{V}^i except the common descendants.$
 - **else** // Insert skylines into \mathcal{Z} , and move on next precedence subset.
 - $\mathcal{Z}.insert(SKY(\mathcal{V})) // Move on superset \mathcal{U} of the last subset \mathcal{V} having no child.$
 - **if** \mathcal{V} is the last subset having no child of \mathcal{U} **then**
 - Call some deterministic tie breaker *e.g.*, object ID.
- **else if** $|SKY(\mathcal{W})| < k$ **then**
 - $\mathcal{Z}.insert(SKY(\mathcal{W}))$
 - $Telescope(\mathcal{S} - \mathcal{Z}, \mathcal{W}, k - |\mathcal{Z}|)$
- **else**
 - Terminate.

Fig. 2. Algorithm *Telescope*

Theorem 3 (Correctness of Algorithm *Telescope*). *For user preference \mathcal{W} and retrieval size k , Algorithm *Telescope* returns correct k results such that, each object in the output queue is (a) a part of skylines and (b) outranks the objects yet to be retrieved, at any point during execution.*

Proof. Immediate from Theorems 1 and 2, every node in the lattice contains only the skyline objects. Further, after the graph transformation in Definition 3, Algorithm *Telescope* ensures no unseen object, yet to be accessed, outranks the skyline objects found in prior by design (as discussed before).

Theorem 4 (Optimality of Algorithm *Telescope*). *After the transformation in Definition 3, Algorithm *Telescope* only visits the node that is absolutely necessary to identify the top- k skyline objects, for user-specified preference \mathcal{W} and retrieval size k . The worst number visited of Algorithm *Telescope* is $O(m)$ which is minimal traversal to obtain the top- k skylines.*

Due to the space limitation, we leave the proof to our extended report [14] and only report the proof sketch for this theorem. As discussed for the correctness proof, Algorithm *Telescope* ensures that no unseen skyline outranks the ones accessed in prior. Further, using proof by contradiction, we can claim Algorithm *Telescope* terminates after visiting only the absolutely necessary nodes for correctness, and nothing else. Summing up, Algorithm *Telescope* is provably correct and optimal for the user-specified preference \mathcal{W} and retrieval size k .

5 Experiments

This section reports our experimental results to validate the effectiveness and efficiency over our Algorithm *Telescope*. First, to validate effectiveness using real-life data, Section 5.1 reports our evaluations over real-life NBA player statistics. Second, to validate efficiency in extensive problem settings, Section 5.2 reports our evaluations over synthetic data of varying problem settings. Our experiments was carried out on a Intel(R) Xeon(TM) machine with 3.20 GHz dual processors and 1GB RAM running LINUX. Algorithm *Telescope* was implemented in C++ language.

5.1 Real-Life Data Set

In this section, we validate the effectiveness of Algorithm *Telescope* by the quality of the skyline objects retrieved from the real-life data. In particular, we use NBA dataset (available from www.nba.com), resulting in 19112 players with 16 numeric attributes including game points, number of rebounds, assists, steals, and blocks. We evaluate with a scenario of identifying $k = 10$ skyline objects over user-specified preferences on three dimensions, which is a useful query for a basketball coach in our illustrative example in Example 2. For instance, when the coach needs to recruit a guard, he may need to focus more on the number of assists or steals, than the number of rebounds or blocks. Similarly, when the coach recruits a center, he may focus on the skylines over the number of rebounds or blocks.

Table 3. Skylines for guard and center positions

Preference: Assists > Steals > Points		Preference: Rebounds > Blocks > Points	
Position	Player	Position	Player
C	Wilt Chamberlain 1961	C	Wilt Chamberlain 1960
C	Wilt Chamberlain 1962	C	Wilt Chamberlain 1961
C	Wilt Chamberlain 1963	C	Artis Gilmore 1971
G	Nate Archibald 1972	C	Artis Gilmore 1973
G	Don Buse 1975	C	Bob Mcadoo 1974
G	Michealray Richardson 1979	C	Kareem Abdul-jabbar 1975
G	John Stockton 1987	C	Mark Eaton 1984
G	John Stockton 1988	G	Michael Jordan 1986
G	John Stockton 1990	G	Michael Jordan 1987
G	John Stockton 1991	C	Patrick Ewing 1989

We implement the above two scenarios of recruiting good guard candidates (with qualitative preference of assists > steals > points) and good center candidates (with qualitative preference of rebounds > blocks > points) and report the results in Table 3. In contrast to existing skyline ranking, which is user-oblivious, our skyline results effectively adapt to the user-specific needs and identify ideal candidates. Observe from Table 3 that our top-10 results correctly identify legendary NBA guards and centers respectively, except all-round players such as Wilt Chamberlain or Michael Jordan who happen to play the roles of both a guard and a center.

5.2 Synthetic Data Set

With the effectiveness of our Algorithm *Telescope* illustrated in Section 5.1, this section discusses the efficiency of our Algorithm *Telescope* over synthetic data of varying problem settings, such as the user preference dimensionality m , data size $|\mathcal{S}|$, and retrieval size k .

More specifically, we randomly generate synthetic data with and without correlations, which is a deciding factor for the number of skylines. Intuitively, if data dimensions are anti-correlated, the number of skylines explodes, as even a tuple dominated in some dimension A is likely to be superior in another dimension B , anti-correlated to A . We thus generate synthetic data using uniform random generator for all dimensions (for **Independent** dataset). In addition, for correlated data, we synthetically introduce correlation by randomly generating $t_i(d_1)$ and generating the rest, *i.e.*, $t_i(d_j)$ for $j > 1$ within the range of $t_i(d_{j-1}) \pm \alpha$ (for **Correlated** dataset) and $-t_i(d_{j-1}) \pm \alpha$ (for **Anti-correlated** dataset). In Fig. 3(a), (b), and (c) report the response time of Algorithm *Telescope* for the above three datasets, over varying dimensionality m , data size $|\mathcal{S}|$, and retrieval size k respectively. Besides, Fig. 3(d) reports the number of the visited nodes in the lattice graph and the left-skewed graph with the datasets.

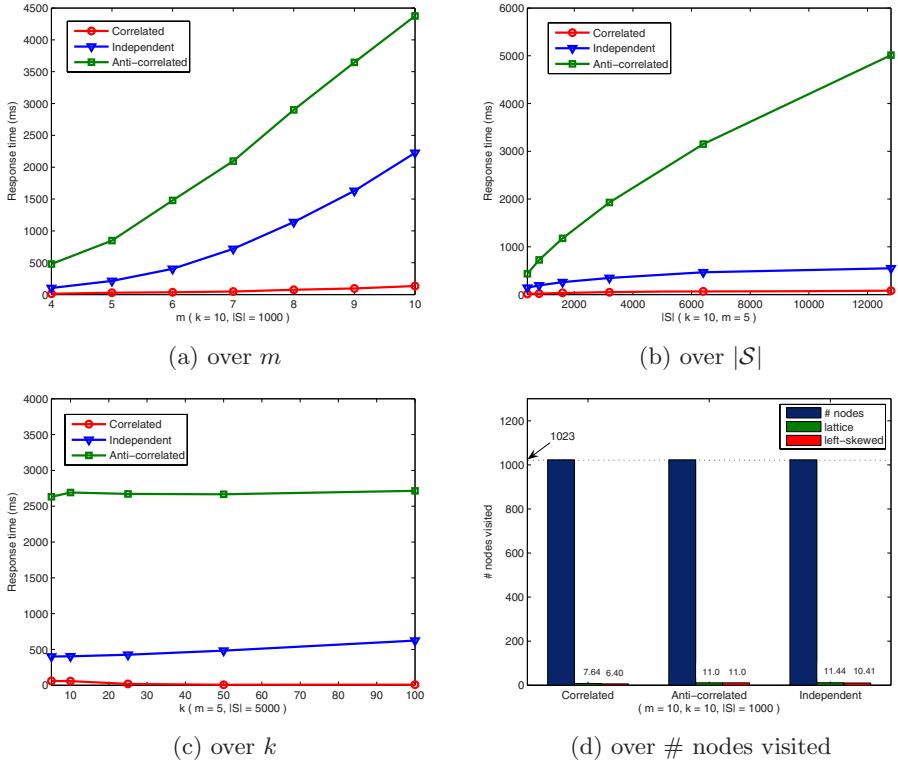


Fig. 3. Response time / The number of nodes visited

Fig. 3(a) reports performance results over varying dimensionality m . Observe that, Algorithm *Telescope* by only visiting the provably minimal number of nodes in the left-skewed graph, scales gracefully over m in all datasets, while the number of nodes to visit explodes exponentially over m , *i.e.*, $O(2^m)$. The performance degradation of anti-correlated dataset can be best explained by the explosion of the number of skylines.

We similarly report our scalability results over cardinality and retrieval size in Fig. 3(b) and (c) respectively. Observe that, in all settings, Algorithm *Telescope* ensures high scalability and low response time *e.g.*, less than 0.6×10^{-2} second in all evaluations. In addition, Fig. 3(d) reports the efficiency of pruning in Algorithm *Telescope*. Observe that Algorithm *Telescope*, by using the lattice graph in Fig. II(left), only visits approximately 1 % of all nodes. Further, by transforming the lattice graph into the left-skewed graph in Fig. II(right), we can further reduce the number of nodes visited by 17 %.

6 Conclusion

This paper studies how to alleviate the curse of dimensionality problem in skyline queries. More specifically, Algorithm *Telescope* zooms into truly interesting

skyline objects, guided by user-specified qualitative preference and retrieval size, which complements existing works that either require users to formulate a rather cumbersome query or do not adapt to user-specific information needs. In particular, Algorithm *Telescope* abstracts skyline ranking as a dynamic search over skyline subspaces to efficiently and effectively identify truly interesting objects for a specific user. Our extensive performance study validates both the effectiveness and efficiency of Algorithm *Telescope* on real-life and synthetic data.

References

1. Yidong Yuan, Xuemin Lin, Qing Liu, Wei Wang, Jeffery Xu Yu, and Qing Zhang. Efficient computation of the skyline cube. In *VLDB 2005*, 2005.
2. Jian Pei, Wen Jin, Martin Ester, and Yufei Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *VLDB 2005*, 2005.
3. Chee-Yong Chan, H.V. Jagadish, Anthony K.H. Tung Kian-Lee Tan, and Zhenjie Zhang. On high dimensional skylines. In *EDBT 2006*, 2006.
4. Chee-Yong Chan, H.V. Jagadish, Kian-Lee Tan, Anthony K.H. Tung, and Zhenjie Zhang. Finding k-dominant skyline in high dimensional space. In *SIGMOD 2006*, 2006.
5. Ronald Fagin. Combining fuzzy information from multiple systems. In *PODS 1996*, pages 216–226, 1996.
6. Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS 2001*, 2001.
7. Nicolas Bruno, Luis Gravano, and Amelie Marian. Evaluating top-k queries over web-accessible databases. In *ICDE 2002*, 2002.
8. Kevin C. Chang and Seung-won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *SIGMOD 2002*, pages 346–357, 2002.
9. Seung-won Hwang and Kevin C. Chang. Optimizing access cost for top-k queries over web sources. In *ICDE 2005*, 2005.
10. Hwanjo Yu, Seung won Hwang, and Kevin Chen-Chuan Chang. RankFP: A framework for supporting rank formulation and processing. In *ICDE 2005*, 2005.
11. Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE 2001*, 2001.
12. Donald Kossmann. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB 2002*, 2002.
13. Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting. In *ICDE 2003*, 2003.
14. Jongwuk Lee, Gae-won You, and Seung-won Hwang. Telescope: Zooming to interesting skylines. In *POSTECH Technical Report*, 2006.

Eliciting Matters – Controlling Skyline Sizes by Incremental Integration of User Preferences

Wolf-Tilo Balke¹, Ulrich Guntzer², and Christoph Lofi¹

¹ L3S Research Center, Leibniz University Hannover
Appelstr 4, 30167 Hannover, Germany
{balke, lofi}@l3s.de

² Institute of Computer Science, University of Tübingen
Sand 13, 72076 Tübingen, Germany
ulrich.guntzer@informatik.uni-tuebingen.de

Abstract. Today, result sets of skyline queries are unmanageable due to their exponential growth with the number of query predicates. In this paper we discuss the incremental re-computation of skylines based on additional information elicited from the user. Extending the traditional case of totally ordered domains, we consider preferences in their most general form as strict partial orders of attribute values. After getting an initial skyline set our basic approach aims at interactively increasing the system’s information about the user’s wishes explicitly including indifferences. The additional knowledge then is incorporated into the preference information and constantly reduces skyline sizes. In fact, our approach even allows users to specify trade-offs between different query predicates, thus effectively decreasing the query dimensionality. We give theoretical proof for the soundness and consistence of the extended preference information and an extensive experimental evaluation of the efficiency of our approach. On average, skyline sizes can be considerably decreased in each elicitation step.

Keywords: skyline queries, partial order preferences, personalization.

1 Introduction

The problem that users cannot sensibly specify weightings or optimization functions for utility assessment of retrieval results has been considered for quite some time in the area of top-k queries and cooperative retrieval. Recently, the novel paradigm of skyline queries [6, 16, 15] has been proposed as a possible (if somewhat incomplete) answer. Skyline queries offer *user-centered* querying as the user just has to specify the basic predicates to be queried and in return retrieves the Pareto-optimal result set. In this set *all possible* best objects (where ‘best’ refers to being optimal with respect to any monotonic optimization function) are returned. Hence, a user cannot miss any important answer. However, this advantage of intuitive query formulation comes at a price: on one hand skylines are rather expensive to compute, on the other hand skylines are known to grow exponentially in size with an increasing number of predicate values [5].

In fact, experiments in [3] show that with as little as 5-6 independent query predicates usually already about 50 % of all database objects have to be returned as the skyline; clearly a prohibitive characteristic for practical uses. The problem even becomes harder if instead of totally ordered domains, partial order preferences on categorical domains are considered. In database retrieval, preferences are usually understood as partial orders [9, 13, 1] of domain values that allow for incomparability between attributes. This incomparability is reflected in the respective skyline sizes that are generally much bigger than in the totally ordered case. On the other hand such attribute-based domains like colors, book titles, or document formats play an important role in practical applications, e.g., digital libraries or e-commerce applications. As a general rule of thumb it can be stated that the more preference information (including its transitive implications) is given by the user with respect to each predicate, the smaller the average skyline set can be expected to be.

Building on our work in [2] in this paper we will discuss the incremental change of skyline sizes based on the newly elicited user preferences. Seeing preferences in their most general form as partial orders between domain values, this explicitly includes the case of totally ordered domains. After getting an (usually too big) initial skyline set our basic approach aims at interactively increasing the system's information about the user's wishes. The additional knowledge then is incorporated into the preference information and helps to reduce skyline sets. Our contribution thus is threefold:

- Users are enabled to specify *additional preference information* (in the sense of domination), as well as *equivalences* (in the sense of indifference) between attributes leading to an incremental reduction of the skyline. Here our system will efficiently support the user by automatically taking care that newly specified preferences and equivalences will never violate the consistency of the previously stated preferences (i.e. users will not encounter conflicts).
- Our skyline evaluation algorithm will allow specifying such additional information *within a certain predicate*. That means that more preference information about a predicate is elicited from the user. Thus the respective preference will be more complete and skylines will usually become smaller. This can reduce skylines to the (on average considerably smaller) sizes of total order skyline sizes by canceling out incomparability between attribute values.
- In addition, our evaluation algorithm will also allow specifying additional relations between preferences on *different predicates*. This feature allows defining the qualitative importance or equivalence of attributes in different domains and thus forms a good tool to compare the respective utility or desirability of certain attribute values. The user can thus express trade-offs or compromises he/she is willing to take and also can adjust imbalances between fine-grained and coarse preference specifications.

Especially the last contribution is of utmost importance and has not been considered in skyline query processing so far. It is the only way – short of dropping entire query predicates – to reduce the dimensionality of the skyline computation and thus severely reduce skyline sizes. Nevertheless the user stays in full control of the information specified and all information is only added in a qualitative way, and not by unintuitive weightings. We will prove in our experiments that using elicited preference information does indeed lead to the expected positive effect on skyline sizes.

2 A Skyline Query Use-Case and Related Work

To introduce the basic concepts of incremental preference enhancement, first we will present a short use case that will serve as a running example throughout the paper.

2.1 Basic Concepts of Partial Order Skyline Processing

Example. Consider a user deciding to buy a car. Usually he/she has preferences on at least some typical attributes like the car type, the color, the price, etc. Figure 1 shows three such preferences in the form of *strict partial orders*. These preferences can either be explicitly provided by the user together with the query or – what is more often the case – are provided as part of a user profile e.g., from typical usage patterns or previous user interactions. Sometimes they are also application/domain inherent like for example the preference on a lowest possible price for articles with the same characteristics in other respects. The skyline is then computed as the *Pareto-optimal set* over these preferences, e.g. a cheap red roadster dominates all expensive red, yellow or green car types, but for instance does not dominate any black car.

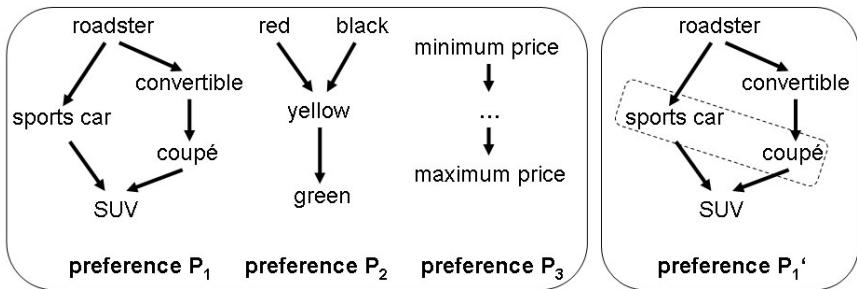


Fig. 1. Three typical user preferences (left) and an enhanced preference (right)

Unlike for example the price preference that adheres to a naturally induced total order, preferences on categorical attributes will usually form only partial orders, expressing a user's indifference or indecisiveness. However, especially these attributes will increase skyline sizes, since the attribute's incomparability demands that they may all be part of the skyline. In fact tests in [2] show that partial order skylines sizes on a set of attribute values are on average about two orders of magnitude bigger than skylines where some total order has been declared on the same set of attributes. For instance a skyline over the preferences in figure 1 would contain all best red cars, as well as the best black cars. If a result size is too large to be manageable by the user, more specific information is required and has to be elicited.

Example (cont.). To reduce skyline sizes indifference can be reduced within each query predicate. A user can explicitly decide to *add a preference* to the current preference relation of a query predicate. For instance a user might state that he/she would rather have a black car than a red car and thus preference P_2 in figure 1 would be transformed into a total order by incrementing the object relationships in the already

known preference relation by the relationships stating that black cars are generally preferred over all red cars with in all other respects equal or worse attribute values.

On the other hand a user might rather want to *state equivalences between attribute values*. Considering preference P_1 the user might express the equivalence between the sports car and the coupé like shown in preference P_1' . Implicitly this equivalence means that both car types are equally desirable and this also has consequences for the induced preference relation. For instance, the preference for convertible car types over coupés now also should imply a preference of convertible car types over sports cars. Stating the equivalence thus allows the user to express that sports cars and coupés are understood as indifferent choices, whereas the choice for a car type with removable top (such as a roadster or convertible) takes precedence for this user.

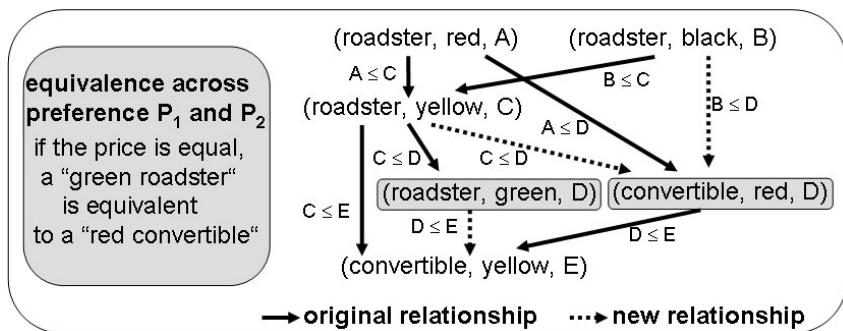


Fig. 2. Original and induced domination relationships on object level

However, a user may not only have a feeling for relationships within a predicate, but also a feeling for the trade-offs he/she is willing to consider. The Pareto order describes the order of all possible ‘packages’ of predicate, i.e. induces an order between value tuples that are represented by at least one database object. Equivalences can be stated with respect to individual pairs of preferences, thus amalgamating preferences and effectively reducing the dimensionality of the skyline query.

Example (cont.). Declaring equivalences between different preferences is especially useful for stating differences in the amount of relaxation between preferences. Figure 2 shows the basic concept for our example, where every database object is a 3-tuple of car type, color and price. For example a user might find a relaxation of his/her color preferences less severe than a relaxation of the respective car’s type. Consider for instance the roasters and convertibles. He/she could consider a green roadster (worst color) as equivalent to a red convertible (best color). The right hand side of Figure 2 shows the old and new domination relationships of different roasters to some convertibles. Note that after introducing the new equivalence all roadsters are considered better than convertibles (given that also the price is better or at least the same). We see the original domination relationships as defined by the Pareto order (black arrows) and those that were newly induced by the stated equivalence (dotted arrows). For example (given a better price) a previously incomparable black roadster now can

be considered better than a red convertible, because it is better than a yellow roadster, which in turn is better than a green roadster that is considered equivalent to the red convertible.

Please note that such equivalences do not always have to lead to a ‘lexicographical’ ordering between preferences, but can also express more fine-grained relations between individual preferences, e.g. a certain amount in price is deemed to make up for one relaxation step in color. In any case, by eliciting new preferences or equivalences, the skyline size can never be increased. If any of the preference relations is enhanced by consistently adding more preference information, more domination relationships are possible in the Pareto order that is used for skyline evaluation. Hence the skyline size is bound to decrease monotonically.

2.2 Related Work

The problematic practical applicability of the skyline query paradigm in the face of exponentially growing result set sizes has been identified soon after its conception [1, 11]. To deal with this serious shortcoming several approaches have proposed the exploration of skylines in the form of user interaction. Since deriving a representative sample was proven NP-hard [14], this is done either by precomputing a ‘skycube’ that allows for OLAP-style interaction, or by exploiting user feedback on skyline samples to restrict the space of possible optimization functions. The latter approach [4, 3] aims at calculating a representative, yet manageable sample of the skyline to derive suitable utility functions for the user. Using these utility functions a top-k based approach can be performed that retrieves a manageable set of best objects, however, restricted to objects similar to those in the sample. In contrast, the skycube (or skyline cube) approach [20, 17] precomputes the skyline sets for various combinations of predicates such that a user can explore the skyline on-line e.g. by adding, dropping or aggregating predicates and consider the changes in the skyline. The major problem of this approach is the vast amount of expensive precomputations, which have to be repeated in the face of update operation to the data, see e.g. [19] for a discussion.

In human-computer interaction and AI, the importance of preference elicitation for a cooperative system behavior has already since long been recognized. Current approaches can be divided into those focusing on structural assumptions and those using feedback of users [21]. The first group features methods like value function elicitation [12] or the analytic hierarchy process [18]. Generally speaking they aim at composing utility value functions to rank a set of alternative choices. Assuming additive independence of predicates each individual predicate’s utility is handled and then composed into a multi-dimensional utility function. A more flexible approach are CP-Networks [7] where the additive independence is replaced by conditional preferential independence allowing to use a set of totally ordered preference relations depending on the objects predicate values. Moreover, statistical approaches for eliciting preferences have been considered [8] where the elicitation process is modeled using a Markov-decision process over possible utility functions. In comparison, the approach in this paper is more general as it does not compose utility functions, but uses partial-ordered preferences that might even include several individual predicates.

Closest to our approach is the work in [10] examining theoretical properties of general incremental elicitation of partial and total order preferences. However, the

work only examines possible preference collisions when combining (incrementally) or revising preferences in query modification and query evaluation.

3 Formalization of the Incremental Skyline Computation

To facilitate the incremental computation of skylines we need to formalize the preference and equivalence information that is exploited to calculate the respective skylines. Given a set of database objects O the preference relation stating the basic set of domination relationships (or for short: preferences) between individual database objects will be denoted as $P \subseteq O^2$. We will assume P to be free of cycles (i.e. consistent) and to induce a partial order between database objects: $\forall x, y \in O$ the expression $(x, y) \in P$ (or alternatively $x <_P y$) will denote that object y dominates object x with respect to all query predicates in the sense of Pareto optimality.

Similarly we will define $Q \subseteq O^2$ as an equivalence relation, i.e. a set of equivalences between database objects such that

- a) Q is an equivalence relation (especially: is symmetric)
- b) $Q \cap P = \emptyset$ (i.e. no equivalence in Q contradicts any strict preference in P)
- c) $P \circ Q = Q \circ P = P$ (i.e. the domination relationships expressed transitively using P and Q should always already be contained in P)

We will call conditions a) to c) the *compatibility of equivalence relation Q with preference relation P* and use this characteristic to avoid inconsistencies between P and Q . Whereas it obviously does not make sense to specify equivalences that do not define an equivalence relation or directly contradict previously specified preference information, condition c) will have to be actively upheld by our incremental skyline computation algorithm. The idea of c) is that we start with only exact value equalities as equivalences (thus c) is trivially true) and then change Q and P accordingly for any equivalences that have been additionally specified.

Definition 1. (Expanded Preference and Equivalence Set)

Let O be a set of database objects, $P \subseteq O^2$ be a strict preference relation, $P^{conv} \subseteq O^2$ be the set of converse preferences with respect to P , and $Q \subseteq O^2$ be an equivalence relation that is compatible with P . Let further $S \subseteq O^2$ be a set of object pairs (called incremental preferences) such that

$$\forall x, y \in O: (x, y) \in S \Rightarrow (y, x) \notin S \text{ and } S \cap (P \cup P^{conv} \cup Q) = \emptyset$$

and let $E \subseteq O^2$ be a set of object pairs (called incremental equivalences) such that

$$\forall (x, y) \in E: (x, y) \in E \Rightarrow (y, x) \in E \text{ and } E \cap (P \cup P^{conv} \cup Q \cup S) = \emptyset.$$

Then we will define T as the transitive closure $T := (P \cup Q \cup S \cup E)^+$ and the expanded preference relation P^* and the expanded equivalence relation Q^* as

$$P^* := \{ (x, y) \in T \mid (y, x) \notin T \} \quad \text{and}$$

$$Q^* := \{ (x, y) \in T \mid (y, x) \in T \}$$

The basic intuition is that S and E contain the new preferences and equivalences that have been elicited from the user additionally to those given in P and Q . The only conditions on S and E are that they can neither directly contradict each other, nor are they allowed to contradict already known information. The sets P^* and Q^* then are the new preference/equivalence sets that incorporate all the information from S and E .

and that will be used to calculate the reduced skyline set. Definition 1 indeed results in the desired incremental skyline set as we will prove in theorem 1:

Theorem 1. (Correct Incremental Skyline Evaluation with P^* and Q^*)

Let P^* and Q^* be defined like in definition 1. Then the following statements hold:

- 1) P^* defines a strict partial order (specifically: P^* does not contain cycles)
- 2) Q^* is a compatible equivalence relation with preference relation P^*
- 3) $Q \cup E \subseteq Q^*$
- 4) The following statements are equivalent
 - a) $P \cup S \subseteq P^*$
 - b) $P^* \cap (P \cup S)^{conv} = \emptyset$ and $Q^* \cap (P \cup S)^{conv} = \emptyset$
 - c) No cycle in $(P \cup Q \cup S \cup E)$ contains an element from $(P \cup S)$

and from either one of these statements follows: $Q^* = (Q \cup E)^+$

Proof:

Let us first show two short lemmas:

Lemma 1: $T \circ P^* \subseteq P^*$

Proof: Due to T 's transitivity $T \circ P^* \subseteq T \circ T \subseteq T$ holds. If there would exist objects $x, y, z \in O$ with $(x, y) \in T, (y, z) \in P^*$, but $(x, z) \notin P^*$, then follows $(x, z) \in Q^*$ because T is transitive and the disjoint union of P^* and Q^* . Due to Q^* 's symmetry we also get $(z, x) \in Q^*$ and thus $(z, y) = (z, x) \circ (x, y) \in T \circ T \subseteq T$. Hence we have $(y, z), (z, y) \in T \Rightarrow (y, z) \in Q^*$ in contradiction to $(y, z) \in P^*$. ■

Lemma 2: $P^* \circ T \subseteq P^*$

Proof: analogous to lemma 1 ■

ad 1) From lemma 1 directly follows $P^* \circ P^* \subseteq P^*$ and thus P^* is transitive. Since by definition 1 P^* is also anti-symmetric and irreflexive, P^* defines a strict partial order. ■

ad 2) We have to show the three conditions for compatibility:

a) Q^* is an equivalence relation. This can be shown as follows: Q^* is symmetric by definition, is transitive because T is transitive, and is reflexive because $Q \subseteq T$ and trivially all pairs $(q, q) \in Q$.

b) $Q^* \cap P^* = \emptyset$ is true by definition 1

c) From lemma 1 we get $Q^* \circ P^* \subseteq P^*$ and due to Q^* being reflexive also $P^* \subseteq Q^* \circ P^*$. Thus $P^* = Q^* \circ P^*$. Analogously we get $P^* \circ Q^* = P^*$ from lemma 2.

Since a), b) and c) hold, equivalence relation Q^* is compatible to P^* . ■

ad 3) Since $Q \subseteq T$ and Q is symmetric, $Q \subseteq Q^*$. Analogously $E \subseteq T$ and E is symmetric, $E \subseteq Q^*$. Thus, $Q \cup E \subseteq Q^*$. ■

ad 4) We have to show three implications for the equivalence of a), b) and c):

a) \Rightarrow c): Assume there would exist a cycle $(x_0, x_1) \circ \dots \circ (x_{n-1}, x_n)$ with $x_0 = x_n$ and edges from $(P \cup Q \cup S \cup E)$ where at least one edge is from $P \cup S$, further assume

without loss of generality $(x_0, x_1) \in P \cup S$. We know $(x_2, x_n) \in T$ and $(x_1, x_0) \in T$, therefore $(x_0, x_1) \in Q^*$ and $(x_0, x_1) \notin P^*$. Thus, the statement $P \cup S \subseteq P^*$ cannot hold in contradiction to a).

c) \Rightarrow b): We have to show $T \cap (P \cup S)^{conv} = \emptyset$. Assume there would exist $(x_0, x_1) \circ \dots \circ (x_{n-1}, x_n) \in (P \cup S)^{conv}$ with $(x_{i-1}, x_i) \in (P \cup Q \cup S \cup E)$ for $1 \leq i \leq n$. Because of $(x_0, x_n) \in (P \cup S)^{conv}$ follows $(x_n, x_0) \in P \cup S$ and thus $(x_0, x_1) \circ \dots \circ (x_{n-1}, x_n)$ would have been a cycle in $(P \cup Q \cup S \cup E)$ with at least one edge from P or S , which is a contradiction to c).

b) \Rightarrow a): If the statement $P \cup S \subseteq P^*$ would not hold, there would be x and y with $(x, y) \in P \cup S$, but $(x, y) \notin P^*$. Since $(x, y) \in T$, it would follow $(x, y) \in Q^*$. But then also $(y, x) \in Q^* \cap (P \cup S)^{conv}$ would hold, which is a contradiction to b).

This completes the equivalence of the three conditions now we have to show that from any of we can deduce $Q^* = (Q \cup E)^+$. Let us assume condition c) holds.

First we show $Q^* \subseteq (Q \cup E)^+$. Let $(x, y) \in Q^*$, then also $(y, x) \in Q^*$. Thus we have two representations $(x, y) = (x_0, x_1) \circ \dots \circ (x_{n-1}, x_n)$ and $(y, x) = (y_0, y_1) \circ \dots \circ (y_{m-1}, y_m)$, where all edges are in $(P \cup Q \cup S \cup E)$ and $x_n = y = y_0$ and $x_0 = x = y_m$. If both representations are concatenated, a cycle is formed with edges from $(P \cup Q \cup S \cup E)$. Using condition c) we know that none of these edges can be in $P \cup S$. Thus, $(x, y) \in (Q \cup E)^+$.

The inclusion $Q^* \supseteq (Q \cup E)^+$ holds trivially due to $(Q \cup E)^+ \subseteq T$ and $(Q \cup E)^+$ is symmetric, since both Q and E are symmetric. ■

The evaluation of skylines thus comes down to calculating P^* and Q^* as given by definition 1 after we have checked their consistency as described in theorem 1, i.e. verified that no inconsistent information has been added. It is a nice advantage of our system that at any point we can incrementally check the applicability and then accept or reject a statement elicited from the user or a different source like e.g. profile information. Therefore, skyline computation and preference elicitation are interleaved in a transparent process.

For the actual skyline computation we rely on the algorithm given in [2] for customized Pareto aggregation. The customized Pareto operator already uses both preference and equivalence information for each predicate. The preference and equivalence information in our case is given by P^* and Q^* respectively.

Only for the predicates spanning across predicates we have to slightly adapt the customized Pareto aggregation. In case of a preference/equivalence connecting two preferences P_i and P_j and/or their respective equivalence sets Q_i and Q_j , we first amalgamate the two individual predicates by customized Pareto aggregation to a new preference P and equivalence set Q as follows (in the notation of [2]): $P := \text{Pareto}(O, P_i, P_j, Q_i, Q_j)$ and $Q := \{(x_1, x_2), (y_1, y_2) \mid (x_1, y_1) \in Q_i \text{ and } (x_2, y_2) \in Q_j\}$

After we have amalgamated the preferences as shown, we can easily insert all preference and equivalence information spanning both predicates and run the normal customized Pareto aggregation for skyline evaluation, however, with the advantage of reduced dimensionality. Moreover, since the new skylines will only get smaller we can restrict all incremental skyline computations to the already retrieved set in the previous step. Thus, the skyline is only once computed expensively over the full database and all subsequent steps are then only calculated over increasingly smaller data sets.

4 Experimental Section

In this section we evaluate the effects and implications of our approach. For a fair comparison several synthetic datasets and preference relations are generated randomly for each measurement series and the averages are reported. Throughout the tests random preferences mimicking realistic preference graphs are successively extended by pieces of equivalence information (thus introducing some new preference relations in P^* and its transitive closure). We evaluate multiple scenarios with changing parameters to study general characteristics of our approach. The base parameters of each scenario, unless stated differently can be found in Table 1. (cf. experiments in [2]):

Table 1. Base parameters for the evaluation scenarios

Parameter	Value
Database Size	100,000
Distribution	uniform
Number of Query Predicates	6
Predicates' Domain Size (# distinct attribute values)	30
Preference Depth (longest path within graph)	8
Edge Degree (ratio between graph nodes and edges)	1.2
Unconnected Degree (ratio between isolated and connected nodes in graph)	0.05

4.1 Influence of Incrementally Adding Equivalence Edges on the Result Size

In this scenario, we examine the average reduction of skyline size during the incremental addition of edges. Our claim is that adding more and more edges will decrease the size of the resulting skyline set significantly. This is especially true for adding equivalences between different predicates, i.e. amalgamating preferences. For performing this evaluation we considered uniform, normal and Zipf distributions of data. During the course of each run, up to ten valid edges (according to definition 1) are randomly inserted *into* or *between* preference relations (each case separately). After incrementally adding an edge, the resulting skyline size is measured. The resulting average sizes are shown for uniform distribution in Figure 3: the average skyline size was reduced in only ten steps to 73 % of its original size in the case of adding only edges within preference graphs and to 34 % using edges between different preferences. Our experiments for data sets following a Gaussian and Zipf distribution provide similar results and thus confirm them. In the Zipf case (at a skew of 0.7), however the initial skyline was already considerably smaller (about 49.000 compared to 62.000 objects) and hence also the decrease in skyline sizes were less pronounced.

4.2 Examination of the Normalized Result Size Reductions

To quantify the respective decline in skyline sizes we examined the behavior after adding each edge. Obviously, different edges can have a vastly different influence on the size reductions. There are some edges (e.g. between leave nodes) that will not contribute much, whereas other edges (e.g. connecting disjoint parts of a graph near

the root) will be highly beneficial. Hence, this effect has to be studied under some suitable normalization. An obvious normalization that can be easily calculated is the number of edges that an incrementally added edge in a base preference actually causes to be inserted in the transitive closure of P^* and Q^* (which in turn form the base for the new skyline calculation). We thus calculated the decrease in skyline size as percentage of a single edge in the transitive closure. The observed mean value of 0.16 shows that per edge in the transitive close the skyline can be expected to decrease by about 0.16%. With a measured standard deviation of 0.13, however, this value is no adequate tool for predicting skyline reductions and a more sophisticated approach, involving more complex statistical characteristics of the data set (cf. e.g. [11]), will be necessary for accurately predicting result skyline reductions.

Therefore, we also checked the impact of new preference information for diverse preferences over the same set of data (i.e. how the actual shape of the preference varies the impact of new information). We measured the average absolute benefit of a single random additional equivalence and considered its distribution. The left hand side of Figure 4 shows our results. The impact of new information shows a mean of about 11247 objects and a standard deviation of 5473. Although it seems to resemble a Gaussian distribution, a Shapiro-Wilk test with a confidence of 95% fails.

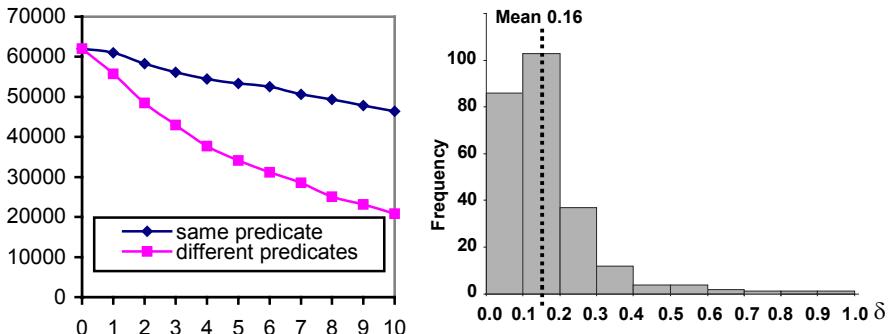


Fig. 3. Result set sizes for incrementally added edges (left) and the frequency histogram of the distribution of the observed normalized size reductions δ (right)

4.3 Influence of Preference Depth on Skyline Result Set Sizes

Finally, we varied the number of additional edges over different preference depths (i.e. approaching a total order). The right hand side of figure 4 reports our results. Plotted are the average respective skyline sizes for 0, 3, 6 and 9 incrementally added edges between preference graphs over a dataset of 100,000 objects. With increasing preference depth, the result set sizes also decrease significantly due to the reduction of incomparable predicate values within the preferences. As the preferences more and more resemble a total order (which is reached at a depth of 30) the initial skyline becomes increasingly lean and the respective reductions by adding more information decrease. Adding more information thus is more important for ‘bushy’ preferences as

opposed to total orders. In any case, also this experiment confirms that eliciting more information from the user leads to significantly diminished skyline sizes.

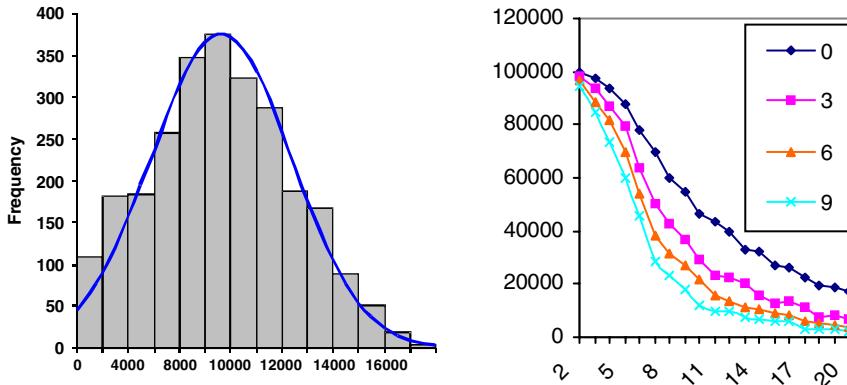


Fig. 4. Absolute impact of new preference information (left) and skyline sizes for varying numbers of additional edges and preference depths

5 Summary and Outlook

In this paper we have shown that unpractical skyline sizes can be controlled by eliciting more information from the user and incrementally recomputing the respective skylines. In our framework users are not only enabled to specify additional preference information (in the sense of domination), but also equivalences (in the sense of indifference) between attributes. Moreover, our skyline evaluation allows for specifying such additional information within a certain predicate and even between preferences on different predicates. In any case users are supported by automatically taking care that newly specified preferences and equivalences will never violate the consistency of any previously stated preferences and their implications. This feature allows defining the qualitative importance or equivalence of attributes in different domains and thus forms a good tool to compare the respective utility or desirability of attribute values: users can express compromises they are willing to take, and adjust imbalances between fine-grained and coarser preference specifications. Our experiments confirm that this indeed can reduce the skylines to the total order skyline sizes by canceling out incomparability and that usually only a few new relations are needed.

Our future work will especially focus on reducing the necessary recomputation steps for deriving the incremental skyline. Since all new information added is only of a local nature, also the new skyline can be expected only to change with respect to several attributes that were affected by the changes. This may lead to considerably reduced computation times for the incremental skyline.

Acknowledgments. Part of this work was supported by a grant of the German Research Foundation (DFG) within the Emmy Noether Program of Excellence.

References

1. W.-T. Balke, U. Güntzer. Multi-objective Query Processing for Database Systems. *Int. Conf. on Very Large Data Bases (VLDB)*, Toronto, Canada, 2004.
2. W.-T. Balke, U. Güntzer, W. Siberski. Exploiting Indifference for Customization of Partial Order Skylines. *Int. Database Engineering and Applications Symp. (IDEAS)*, Delhi, India, 2006.
3. W.-T. Balke, J. Zheng, U. Güntzer. Efficient Distributed Skylining for Web Information Systems. *Int. Conf. on Extending Database Technology (EDBT)*, Heraklion, Greece, 2004.
4. W.-T. Balke, J. Zheng, U. Güntzer. Approaching the Efficient Frontier: Cooperative Database Retrieval Using High-Dimensional Skylines. *Int. Conf. on Database Systems for Advanced Applications (DASFAA)*, Beijing, China, 2005.
5. J. Bentley, H. Kung, M. Schkolnick, C. Thompson. On the Average Number of Maxima in a Set of Vectors and Applications. *Journal of the ACM (JACM)*, vol. 25(4) ACM, 1978.
6. S. Börzsönyi, D. Kossmann, K. Stocker. The Skyline Operator. *Int. Conf. on Data Engineering (ICDE)*, Heidelberg, Germany, 2001.
7. C. Boutilier, R. Brafman, C. Geib, D. Poole. A Constraint-Based Approach to Preference Elicitation and Decision Making. *AAAI Spring Symposium on Qualitative Decision Theory*, Stanford, USA, 1997.
8. C. Boutilier. A POMDP Formulation of Preference Elicitation Problems. *National Conference on Artificial Intelligence (AAAI)*, Edmonton, USA, 2002.
9. J. Chomicki. Preference Formulas in Relational Queries. *ACM Transactions on Database Systems (TODS)*, Vol. 28(4), 2003.
10. J. Chomicki. Iterative Modification and Incremental Evaluation of Preference Queries. *Int. Symp. on Found. of Inf. and Knowledge Systems (FoIKS)*, Budapest, Hungary, 2006.
11. P. Godfrey. Skyline Cardinality for Relational Processing. *Int. Symp. on Foundations of Information and Knowledge Systems (FoIKS)*, Wilhelminenburg Castle, Austria, 2004.
12. R. Keeney, H. Raiffa. Decisions with Multiple Objectives: Preferences and value trade-offs. *Cambridge University Press*, 1976.
13. W. Kießling. Foundations of Preferences in Database Systems. *Int. Conf. on Very Large Databases (VLDB)*, Hong Kong, China, 2002.
14. V. Koltun, C. Papadimitriou. Approximately Dominating Representatives. *Int. Conf. on Database Theory (ICDT)*, Edinburgh, UK, 2005.
15. D. Kossmann, F. Ramsak, S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. *Int. Conf. on Very Large Data Bases (VLDB)*, Hong Kong, China, 2002.
16. D. Papadias, Y. Tao, G. Fu, B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. *Int. Conf. on Management of Data (SIGMOD)*, San Diego, USA, 2003.
17. J. Pei, W. Jin, M. Ester, Y. Tao. Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces. *Int. Conf. on Very Large Databases (VLDB)*, Trondheim, Norway, 2005.
18. T. Satty. A Scaling Method for Priorities in Hierarchical Structures. *Journal of Mathematical Psychology*, 1977.
19. T. Xia, D. Zhang. Refreshing the sky: the compressed skycube with efficient support for frequent updates. *Int. Conf. on Management of Data (SIGMOD)*, Chicago, USA, 2006.
20. Y. Yuan, X. Lin, Q. Liu, W. Wang, J. Yu, Q. Zhang. Efficient Computation of the Skyline Cube. *Int. Conf. on Very Large Databases (VLDB)*, Trondheim, Norway, 2005.
21. L. Chen, P. Pu. Survey of Preference Elicitation Methods. *EPFL Technical Report IC/2004/67*, Lausanne, Swiss, 2004.

Optimizing Moving Queries over Moving Object Data Streams

Dan Lin¹, Bin Cui^{2,*}, and Dongqing Yang²

¹ National University of Singapore

lindan@comp.nus.edu.sg

² Peking University, China

{bin.cui,dqyang}@pku.edu.cn

Abstract. With the increasing in demand on location-based aware services and RFIDs, efficient processing of continuous queries over moving object streams becomes important. In this paper, we propose an efficient in-memory processing of continuous queries on the moving object streams. We model moving objects using function of time and use it in the prediction of usefulness of objects with respect to the continuous queries. To effectively utilize the limited memory, we derive several replacement policies to discard objects that are of no potential interest to the queries and design efficient algorithms with light data structures. Experimental studies are conducted and the results show that our proposed method is both memory and query efficient.

1 Introduction

With rapid advances in electronic miniaturization, wireless communication and position technologies, moving objects that acquire and transmit data are increasing rapidly. This fuels the demand for the location-based services and also deployment of Radio Frequency Identification (RFID) in tracking and inventory management applications. In inventory tracking like applications, disclosure of object positions forms spatio-temporal data streams with high arrival rate, and queries act upon them tend to be continuous and moving. Consequently, queries must be continuously updated and any delay of query response may result in an obsolete answer [7]. Moving object data stream management systems [8] have been designed to handle massive numbers of location-aware moving objects. Such systems receive their input as streams of location updates from the moving objects. These streams are characterized by their high input rate, and they cannot be stored and need to be processed on the fly to answer queries. Clearly, the disk-based structures are not able to support the fast updates and provide quick response time. The PLACE [8] extended data streaming management systems to support location-aware environments. However, the PLACE can only manage the snapshots of objects and queries at each timestamp, which inevitably increases the amount of data information. Additionally, it stores the entire dataset in the server for query processing, which may not be applicable for data stream management system. On the other hand, studies of real positional information obtained from GPS receivers installed in

* Contact author. This work is supported by the NSFC under grant No. 60603045.

cars show that representing positions as linear functions of time reduces the numbers of updates needed to maintain a reasonable precision by as much as a factor of three in comparison to using constant functions [2]. Linear functions are thus much better than constant functions in the data streaming environment.

As with other data streams, processing of continuous spatio-temporal queries over the moving object stream requires the support of in-memory processing. Existing disk-based algorithms cannot be easily turned into in-memory methods, because the underlying structures tend to be bulky and index all data points due to the availability of cheap storage space. Existing tree-based indexing structures [4][10][11][14] for moving objects focus on reducing disk accesses since the execution time is dominated by the I/O operations. In fact, for some indexes, fast retrieval is achieved by preprocessing and optimization before insertion into the index [14]. In this paper, we propose an efficient approach which is able to handle moving objects and queries represented by functions. Due to the limited amount of memory, we design light data structures, based on hash tables and bitmaps. To manage the limited amount of buffer space, we design several replacement policies to discard objects that are of no potential interest to the queries. Experimental results demonstrate that our algorithms can achieve fast response time and high accuracy with a small memory requirement.

The rest of the papers is organized as follows. Section 2 defines the problem and reviews the related work. Section 3 introduces the overall mechanism. Section 4 presents the algorithms of continuous range queries. In section 5, we report the experimental results. Finally, Section 6 concludes with the paper.

2 Problem Statement and Related Work

2.1 Problem Statement

The moving object data stream is made up of a sequence of update information of moving objects. We assume that moving objects are capable of repeatedly transmitting their positions and velocities to a central server. Then, each tuple in the stream includes $\langle OID, \overline{Op}, \overline{Ov}, Ot \rangle$, where OID is the object ID, Ot is the update time, \overline{Op} and \overline{Ov} are the position and velocity at time Ot respectively. The incoming tuple is the update information of the existing tuple with the same OID . To reduce the update frequency, a linear function is used to model the trajectory of a moving object. A moving object is required to transmit a new location to the server when the deviation between its real location and its server-side location exceeds a threshold, dictated by the services to be supported. In keeping with this, we define the *maximum update time* (U) as a problem parameter. This quantity denotes the maximum time duration in-between two updates of the position of any moving object.

The query data stream is comprised of a sequence of two types of queries: continuous static range query and continuous moving range query. They are defined as follows:

- *Continuous static range query*: Given a static range R at time Qt , the query needs continuously reporting all the moving objects within the range R from time Qt .
- *Continuous moving range queries*: Given a range R moving at the velocity \overline{Qv} , and a time Qt , the query needs continuously reporting all the moving objects inside $R(t)$ from time Qt , where $R(t)$ denotes the range at time t after Qt .

In fact, the static query can be treated as the special case of the moving query where the velocity is equal to zero. Therefore, each tuple in the query data stream can be represented using the same format $\langle QID, R, \overline{Qv}, Qt \rangle$. Similar to the moving object data stream, the newly incoming query will replace the tuples with the same QID in the memory. Without loss of generalization, we consider the square range throughout the paper.

2.2 Related Work

There are numerous work in the area of spatio-temporal query processing on moving objects (e.g., [14][6][12][10][11][14][16]). However, these disk-based approaches may not be suitable for scalable, real-time location based queries because of high I/O costs, even when sophisticated buffer management is employed. Although it is possible to tailor these methods and put the entire data and indexes into the main memory to speed up, this may consume too much memory. In this paper, we have proposed a main memory based index approach. Our approach does not intend to store the data of all moving objects, because only some objects will be included in the queries answers.

In [5][15], continuous range queries are made over moving objects. The queries being considered are however static. In [7][8][9], the problem of moving queries over moving objects are discussed. However, their approaches are to store snapshots of queries and objects at each timestamp making it necessary to store and process these snapshots on the disk. To ensure efficient processing, our work here try to address the same problem using only in-memory processing.

3 Continuous Query Processing on Moving Object Streams

3.1 System Architecture

In this section, we introduce the QMOS (Query Moving Object Stream) system. Figure 1 gives an overview of the QMOS system. There are four in-memory storages (shaded boxes): object pool, query pool, event queue and query filter; and two processors (white boxes): query processor and discarding processor.

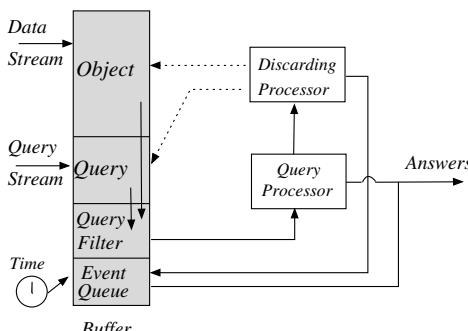


Fig. 1. An Overview of the System Architecture

An incoming object is first put into the *object pool*, and then will be sent to the *query processor* together with selected queries from the *query filter*. The query processor will generate three kinds of results: current answer, potential answer, and none answer. *Current answer* means that the incoming object is one of the answers of the query at the current time, which will be directly reported to the user. *Potential answer* means that the incoming object will be one of the answers of the query at some future time (within the maximum update time interval U). *None answer* means that the incoming object is neither a current answer nor a potential answer. Both of them will be further sent to the *discarding processor*. Since the memory is limited, potential answers and none answers need to be judged whether they are valuable to be stored. The discarding processors will provide a feedback to the object pool if the incoming object can be discarded. Valuable potential answers will then be stored as events in the *event queue*. As time passes, potential answers may turn into current answers and be reported to the users. In addition, the event queue also handles objects which leave the query answer sets. It is worth noting that the query answer set is maintained incrementally. There is an output only when the query result has been changed, due to adding or deleting an object from the answer set.

The processing of an incoming query is relatively simple. If the memory is enough, we store it in the *query pool*, and register its summary information in the *query filter*. Otherwise, the *Discarding processor* is triggered to find out whether there is some space can be used for the new query.

3.2 Storage Components

The *object pool* stores the information of the moving objects. Each tuple in the object pool is in the form of $\langle OID, \overline{Op}, \overline{Ov}, Ot, PA, Ca, Evt \rangle$, where OID , \overline{Op} , \overline{Ov} , Ot are used to represent the object, PA is the number of queries of which the object is a potential answer, Ca is a bit-map storing the entries to the queries of which the object is a current answer, and Evt is also a bit-map used to locate the related events of this object.

The *query pool* stores the information of the queries. Each tuple in the *query pool* consists of $\langle QID, R, Qt, Qa, Evt \rangle$, where QID is the query ID, R is the query range, Qt is the query starting time, Qa is a pointer to the query results, and Evt is used to locate the related events of the query (the same as the corresponding part in the object pool). Further, R is represented by $(\overline{Qp}, \overline{Qv}, L)$, where \overline{Qp} stores the left bottom corner of the query window, \overline{Qv} is the moving velocity of the query window, and L is the length of the query window.

The *event queue* stores future events when an object will join or leave current query answer set. Each tuple consists of four components: t , pO , pQ , and M . t is the time that the event may happen. pO is a pointer to the object stored in the object pool, and pQ refers to the query that this object may affect. M is a one-bit mark: if in the event the object will become one of the query answers, M is set to 1; if the object will no longer be the answer, M is set to 0.

Object pool, query pool and event queue are all organized as hash tables where the keys are OID , QID and t respectively. The lengths of the hash tables are determined by the memory size. The hash structure is preferred over other kinds of data structures

since (i) these data are usually retrieved by their key values and hashing techniques provide fast and direct access; (ii) the memory is limited and hash structures have less storage overhead.

The *query filter* is designed to accelerate the query processing. It is a grid structure which captures the current and future positions of moving queries. Basically, we partition the space into a regular grid where each cell is a bucket. Each bucket contains pointers to the queries passing this bucket.

3.3 Data Processing

We proceed to present how the system manages the two kinds of incoming data (moving objects and queries) and the internal data – events. During all processes, whenever there is not enough memory, discarding policies are applied to remove useless data to collect memory. We defer the discussion of the discarding policies to the next subsection.

- **Moving Object Data Streams**

An incoming object O is processed as follows. First, we check whether the object ID has already existed in the object pool. If yes, we modify the corresponding tuple by using the new information including position \overline{Op} , velocities \overline{Ov} , update time Ot . Information with regards to this object O in the query results and the event queue is removed, since they become obsolete now. Then object O will be computed with queries selected by the query filter. We need to decide whether we store this object in the memory. We will store it only if it proves to be useful. In particular, the coming object is useful if it is a current/ potential answer of a query, and there is enough memory after applying discarding policies. After the object O is successfully stored in the memory, the pointer to the object will be inserted to the query answer set where it is a current answer, and the leaving events and potential answers will be added into the event queue.

The details of query and discarding processes will be addressed later. The deletion and insertion of the object in the object pool is done fast by hashing the OID , similarly to the insertion of the related events and query answers. While the deletion of related events and query answers is a little more complex. The straightforward way is to scan the whole event queue (query pool) to find the events (queries) related to the object, which is obviously inefficient and may result in an unbearable delay when the memory size is large. To avoid such a brute force method, we propose the following techniques.

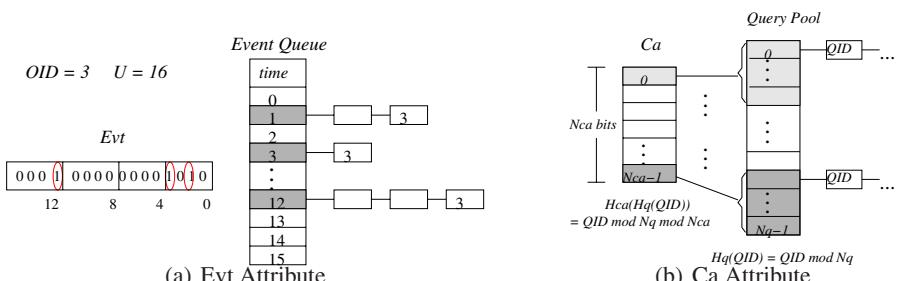


Fig. 2. Examples of Evt and Ca Attributes

The search of the related events is managed with the aid of the *Evt* attribute of the object. Specifically, one bit in the *Evt* is related with one timestamp in the event queue, and the bit will be set to 1 if there is an event with respect to the object happening at the corresponding timestamp. For example (see Figure 2(a)), assuming that $OID = 3$ and $U = 16$, the object has related events at time 1, 3 and 12. Then the 1st, 3rd and 12th bits in the *Evt* are set to 1, others are 0. By checking the *Evt*, we can easily find the entries to the related events of the object and avoid scanning the entire event queue.

The search of the related queries is accelerated by the *Ca* attribute of the object. Different from the *Evt*, the one-one map (i.e. one bit to one entry in the hash table of the query pool) may lead to a long *Ca*, because the number of queries in memory could be large when the memory scales up (i.e. the length of the hash table of the query pool may grow up). Therefore, we employ a second level hashing over the query IDs, where each bit of *Ca* corresponds to a series of entries in the hash table of the query pool. As shown in Figure 2, suppose that the length of the hash table of the query pool is N_q , and the number of the bits in *Ca* is N_{ca} . Queries are first hashed to the hash table of the query pool by the function $H_q(QID) = QID \bmod N_q$. Then the mapping function for the *Ca* is $H_{ca}(QID) = H_{ca}(H_q(QID)) = (QID \bmod N_q) \bmod N_{ca}$.

• Queries

For an incoming query Q , we insert $\langle QID, \overline{Qp}, \overline{Qv}, L, Qt, NULL \rangle$ into the query table to represent the new query. The trajectory of the new query will be registered in the query filter. The new query only considers the objects coming after it, which means it needs some time to “warm up”. The “warming-up” time could be short since objects are updated frequently. If the query expires, we remove the entry from the query pool, and the events related to the old query (the procedure is similar to that in the previous section). Note that objects become none answers after the deletion of the query are automatically discarded from the memory.

• Events

As time passes, the event queue is checked to update current answers of queries. All events whose start time is less than or equal to current time are evaluated. Recall that, the events are stored in a hash table with the length equal to the maximum update interval U . By hashing the current timestamp t , we can find its entry in the hash table.

There are two kinds of events: objects leaving or entering the query range. According to the type of an event, different actions are performed. Given an event $\langle t, pO, pQ, M \rangle$, if the mark M equals to 1, the object pO pointing to should be inserted into the answer list of the corresponding query that pQ points to. If M equals to 0, which means the object O is no longer an answer of the query Q , then O is removed from the answer list of Q . In both situations, the *Ca* and *Evt* attributes of O should be adjusted. Finally, we delete the event itself.

3.4 Discarding Policy

Continuous queries over infinite streams may require infinite working memory. Thus, an essential solution to answer such queries in bounded memory is to discard some unimportant data when the memory is full. Our proposed discarding policies comply with the basic rule that discarding data of lowest priority first. In our scenario, we define

the priorities of the data as that: the query data is most important, followed by the current answer and the potential answer.

Each time the memory is full, we first attempt to discard objects which are neither current answers nor potential answers. If this operation fails, we further apply any of the following three policies.

1. Discard the oldest object according to its insertion time. The idea behind the Policy 1 is that the oldest object has the highest probability to be updated first, and thus the influence of discarding this object may be ended within the shortest time.
2. Discard the object whose first appearance in the event queue is latest than that of any other object and it is an entering event. The motivation of Policy 2 is to keep the query answers unaffected as long as possible. Therefore, it picks the object which is the last one to become a potential answer. Combined with the idea of Policy 1, we may have a variation of Policy 2: discard the object which has the longest time interval between its insertion time and the time it becomes an answer.
3. Discard the object that affects fewest queries. Different from the first two policies that both take into account the time effect, Policy 3 aims to minimize the number of queries that the object affects.

All the policies share the same purpose that minimizes the error rate of the query answers after the discarding. Note that the query data will be discarded only when the memory is fully occupied with queries.

Next, we introduce the discarding process. Any policy is realized by scanning the object pool once. Policy 1 compares the insertion time Ot of each object and discards the one with smallest Ot . Policy 2 is done by examining the attribute Evt of an object, where the lowest non-zero bit refers to the first event of the object. We then need to check whether the event is an entering event or a leaving event. For the Policy 3, the number of related queries can be approximated by the sum of non-zero bits in Ca and Evt . If the exact number is required, we can further access corresponding tuples in the event queue and query pool according to Ca and Evt .

4 Algorithms of Continuous Range Queries

4.1 Processing a Single Query

In the two-dimensional space, given a continuous range query $\langle QID, \overline{Qp}, \overline{Qv}, L, Qt \rangle$, the query range at time t ($Qt \leq t$) can be represented by the left-bottom and right-top corner, $[(Qp_x + Qv_x(t - Qt), Qp_y + Qv_y(t - Qt)), (\underline{Qp}_x + \overline{Qv}_x(t - Qt) + L, \underline{Qp}_y + \overline{Qv}_y(t - Qt) + L)]$. For an incoming object $\langle OID, Op, Ov, Ot \rangle$, we need to identify whether it is a current answer or a potential answer.

An object is a current answer to the range query if its position at current time t_c is inside the query range at time t_c . We first compute the query range at t_c by its moving function, and then compare the position of the object with the left-bottom and right-top corner of the query range directly.

$$\begin{cases} Qp_x + Qv_x(t_c - Qt) \leq Op_x \leq Qp_x + Qv_x(t_c - Qt) + L \\ Qp_y + Qv_y(t_c - Qt) \leq Op_y \leq Qp_y + Qv_y(t_c - Qt) + L \end{cases}$$

If the above conditions are satisfied, the object is a current answer to the range query and will be added into the answer list. The remaining task is to compute the time it leaves the query range, and insert the leaving event to the event queue. As the object is already inside the query range, its future trajectory will have only one intersection point with the query range, and the intersection time is the leaving time. The details of the computation will be explained shortly.

An object is a potential answer to the range query if its position at future time t_f (not later than the maximum update interval) is inside the query range at time t_f . Then we need to compute the time when the object enters the query range, and insert this future event to the event queue. As the object is currently outside the query range, its future trajectory may have at most two intersection points with the query range. The earlier intersection time is the entering time and the other one is the leaving time.

We proceed to present how to compute the intersection time. Figure 3 shows a continuous range query and an incoming moving object, where the solid rectangle presents the query range at the current time, the rectangles with broken line denotes the query ranges at near future, the black point is the moving object, and the connecting line with arrow shows the object's future trajectory. To check whether the object's future trajectory intersects with the query range, let us consider the four borders of the query range, AB, BC, CD, DA , one by one. The border AB moves at the speed of Qv_x , and thus the line at time t (denoted as L_{ab}) it resides in can be described by the equation: $x = Qp_x + Qv_x(t - Qt)$. If the object's trajectory intersects with AB , it must also intersects with L_{ab} . In other words, the object's x coordinate should be on L_{ab} at the intersection time. Assuming that the intersection time is t_{ab} , we have the equation: $Ov_x + Ov_x(t_{ab} - Qt) = Qp_x + Qv_x(t_{ab} - Qt)$. By solving the equation, we obtain the following results:

$$t_{ab} = \begin{cases} \frac{(Qp_x - Ox) - (Qv_x \cdot Qt - Ov_x \cdot Qt)}{Ov_x - Qv_x}, & Ov_x \neq Qv_x; \\ +\infty, & Ov_x = Qv_x. \end{cases}$$

Note that, when $Ov_x = Qv_x$, i.e. the object and the border AB move at the same speed and same direction, they will never meet each other. Therefore, the t_{ab} is set to be the infinite large $+\infty$ in this case.

The resultant t_{ab} value is invalid if it does not satisfy the constraints: (i) $t_{ab} > Qt$, i.e. the intersection time should be later than the object insertion time; (ii) $t_{ab} > Qt$,

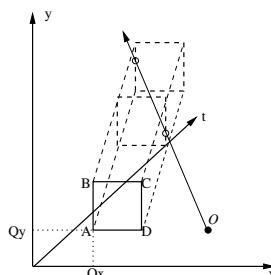


Fig. 3. An Example of a Continuous Range Query

i.e., the intersection time should be later than the query starting time; (iii) $t_{ab} < Ot + U$, i.e., the intersection time should not exceed the validity period of the object. Invalid t_{ab} will also be reset to the infinite large $+\infty$.

So far the t_{ab} we computed is only the intersection time of the object's trajectory and the line that the AB belongs to. We need to further check whether the intersection point lies in the line segment AB . Suppose that the t_{ab} is valid, we can use it to compute the intersection point $P(P_x, P_y)$, where $P_x = Op_x + Ov_x(t_{ab} - Ot)$, and $P_y = Op_y + Ov_y(t_{ab} - Ot)$. Then we compare the y coordinate of P and points A, B . If $A_y \leq P_y \leq B_y$, the intersection point is in the segment AB , which means we obtain one useful intersection time. Otherwise, we again set the t_{ab} to be $+\infty$.

The similar computation is carried out for the other three borders BC, CD and DA . The entering time t_e is the minimum value of the four intersection times, and the leaving time t_l is the finite maximum value of the four intersection times. Note that we may not need to compute the four intersection times. If we have obtained two intersection times which are not $+\infty$, we do not need to process the remaining borders.

4.2 Processing Multiple Queries

An incoming object could be a current or potential answer of multiple queries in the memory. Comparing it with all the queries one by one may result in high query cost. Therefore, we propose a query filter to prune the searching space.

The query filter is a regular grid structure which partition the space into equal cells. Each cell stores pointers to queries which pass by the cell during the maximum update interval U . The pointer to one query may be stored several times in the grid due to the movement of the query. To reduce the number of duplications, we need to decide a reasonable grid cell size. For example, we can set the extent of cell to be slightly larger than $v_{max} \cdot U$, where the v_{max} is the maximum speed of a query. For a query with a long lifetime, we will update its information in the grid every U time interval. Moreover, in order to speed up the mapping process, we do not compute the exact cells that the queries intersects with. Instead, we map the minimum bounding rectangle of the query sweeping region (during U) to the grid as shown in Figure 4(a).

We are now ready to look at how the query filter works. For example, in Figure 4(b), given an incoming object O at time Ot , we will map it to the grid in the similar way as we have done to the query. First we compute its position at time $Ot + U$. The search

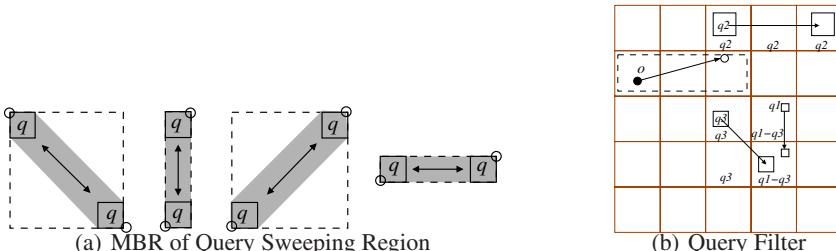


Fig. 4. Query Filter Construction

space (the dashed rectangle in the figure) is the rectangle determined by the two positions at Ot and $Ot + U$ respectively. Then only queries registered inside this rectangle need to be computed.

5 Performance Study

All the experiments are conducted on a 2.6G Hz P4 machine with 1Gbyte of main memory. The memory for our application is limited from 100K to 2M. Moving objects are represented as points in the space domain of 1000×1000 . The datasets were generated by a typical data generator [11]. The maximum interval between two successive updates of an object is equal to 30 time units. Queries follow the same distribution of the moving objects. The moving speed of the queries is half of the speed of objects. The query window size is 0.01% of the space. The number of queries existing at the same time varies from 100 to 500. Unless noted otherwise, we use 300K memory for 100K moving objects when there are 100 queries at each timestamp.

We evaluate the memory requirement, the accuracy and the response time of the proposed three policies. The memory requirement is compared with the B^x -tree. The accuracy function is $Accuracy = \frac{\text{Number of answers produced by the algorithm}}{\text{Number of correct answers}}$. The response time is defined as the time interval between the input of a data and the output of the result regarding to this data.

- **Effect of Memory Size.** The first round of experiments evaluate efficiency of the three discarding policies when varying the total available memory size from 100K bytes to 500K bytes. The number of moving objects are 100K, and the data streams of their update information is of size 217K tuples during 30 timestamps.

Figure 5(a) shows the results of the accuracy at timestamp 30. As shown, the performances of all the policies improve with the increasingly large memory size. The reason is straight forward: larger memory can hold more answers. When the memory size reaches beyond a certain point ($> 300K$), the accuracy of all the policies approaches 100%. Note that 300K is about only 13% of the space used to store all the objects. This is because our algorithm only catches query answers and the result demonstrates its space efficiency. We can also observe that Policy 2 always yields higher accuracy than the other two policies. The reason could be that Policy 2 maximizes the valid period of query results.

Figure 5(b) shows the average response time of the three policies during one maximum update interval. We can see that as the memory size increases, the response time of three policies first increases slightly and then almost keep constant. For an object, the response time is the sum of query processing time and the discarding processing time. The query processing time will not be affected by the memory size when the query number is fixed, and thus the variation of the response time is mainly due to the variation of the discarding processing time. As the memory increases, the time to find a replacement slows down, whereas the need to execute a discarding policy is reduced. When these two factors reach a balancing station, the performance becomes stable. In addition, the resultant three curves are close to one another possibly because that the discarding process is only different in the selection metric, and hence the processing time is similar.

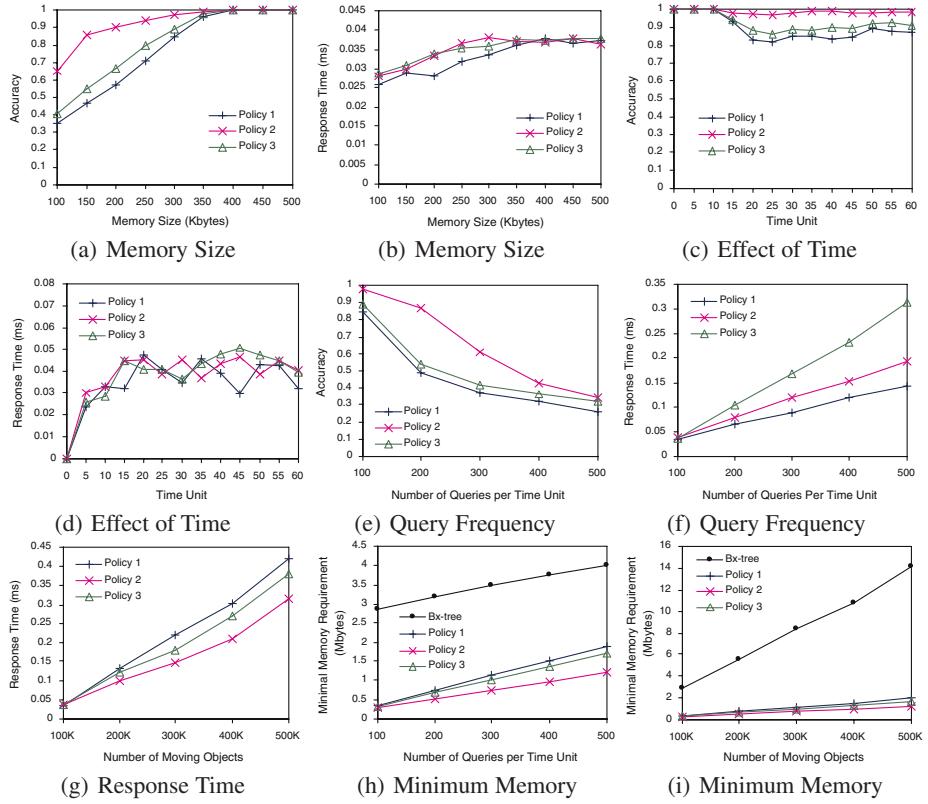


Fig. 5. Experimental Results

- **Effect of Time.** Next, we investigate performance degradation across time. The 100K moving objects are kept updated during 60 timestamps (two times of maximum update interval). As shown in Figure 5(c) and (d), the performance of all the policies decreases a little as time passes, which is due to the execution of the discarding policies, and the larger query range.
- **Effect of Number of Queries Per Time Unit.** In this set of experiments, we vary the number of queries at the same timestamp. We fix the memory size to 300K and test the accuracy and response time. From Figure 5(e) and (f), we observe that the performance degenerates with growing number of queries at the same time. The decrease of the accuracy is mainly caused by the increase of the result dataset. Due to the memory limitation, even potential answers at near future time may be discarded, which affects the accuracy and also increases the processing time.
- **Effect of Data Size.** To study the scalability of our algorithms, we examine the method with varying the number moving objects from 100K to 500K. For the response time (see Figure 5(i)), Policy 2 is the best since it requires the smallest memory so that the discarding process can be executed fastest.

• **Comparison with the B^x -tree.** To show the effectiveness of the proposed method, we compare it with the B^x -tree [4] which has much smaller space requirement compared with other existing index structures, e.g. the TPR*-tree [4]. We first explore the minimum memory required for each policy to achieve high accuracy (above 99%) by varying the numbers of queries per time unit. Note that, the B^x -tree stores all the objects for queries. Figure 5(h) shows the results. Not surprisingly, the minimum memory required for all policies increases with the number of queries. However, our algorithms can save up to 90% space compared with the B^x -tree. Among three policies, Policy 2 has the smallest space requirement, followed by Policy 3 and 1. This is consistent with the previous results in Figure 5(a). Those policies perform better when using the same size of memory, will need less space to reach high accuracy.

Figure 5(i) shows that our algorithms scale very well compared with the B^x -tree for large data sizes. By using our algorithm, less than 2M bytes memory is required for the 500K data, whereas the B^x -tree needs about 15M bytes space.

6 Conclusion

In this paper, we proposed a novel scheme which can handle infinite data streams in memory, and provide prompt response, by compromising with small errors. Our approach supports continuously moving queries over moving objects, both of which are represented by linear functions. Due to the constraints of the memory size and response time, we propose light data structures, and employ hashing techniques. Also, we derive several replacement policies to discard objects that are of no potential interest to the queries. Experimental studies were conducted and the results show that our proposed method is both memory and query efficient.

References

- Y. Chen, F. Rao, X. Yu, D. Liu, and L. Zhang. Managing Location Stream Using Moving Object Database. *Proc. DEXA*, pp. 916–920, 2003.
- A. Čivilis, C. S. Jensen, J. Nenortaitė, and S. Pakalnis. Efficient Tracking of Moving Objects with Precision Guarantees. *Proc. MobiQuitous*, pp. 164–173, 2004.
- H. G. Elmongui, M. Ouzzani and W. G. Aref. Challenges in Spatio-temporal Stream Query Optimization. *Proc. MobiDE*, pp. 27–34, 2006.
- C. S. Jensen, D. Lin and B. C. Ooi. Query and Update Efficient B+-Tree Based Indexing of Moving Objects. *Proc. VLDB*, pp. 768–779, 2004.
- D. V. Kalashnikov, S. Prabhakar, W. G. Aref, and S. E. Hambrusch. Efficient Evaluation of Continuous Range Queries on Moving Objects. *Proc. DEXA*, pp. 731–740, 2002.
- Y. Li, J. Yang, and J. Han. Continuous K-Nearest Neighbor Search for Moving Objects. *Proc. SSDBM*, pp. 123–126, 2004.
- M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. *Proc. SIGMOD*, pp. 623–634, 2004.
- M. F. Mokbel, X. Xiong, M. A. Hammad, and W. G. Aref. Continuous Query Processing of Spatio-temporal Data Streams in PLACE. *Proc. STDBM*, pp. 57–64, 2004.
- R. V. Nehme, and E. A. Rundensteiner. SCUBA: Scalable Cluster-Based Algorithm for Evaluating Continuous Spatio-temporal Queries on Moving Objects. *Proc. EDBT*, pp. 1001–1019, 2006.

10. J. M. Patel, Y. Chen, and V. P. Chakka. STRIPES: An Efficient Index for Predicted Trajectories. *Proc. SIGMOD*, pp. 637–646, 2004.
11. S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. SIGMOD*, pp. 331–342, 2000.
12. D. V. Kalashnikov, S. Prabhakar, and S. E. Hambrusch. Main Memory Evaluation of Monitoring Queries Over Moving Objects. *Distributed and Parallel Databases*, pp. 117–135, 2004.
13. Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. *Proc. VLDB*, pp. 287–298, 2002.
14. Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In *Proc. VLDB*, pp. 790–801, 2003.
15. K. L. Wu, S. K. Chen, and P. S. Yu. Indexing continual Range Queries with Covering Tiles for Fast Locating of Moving Objects. *Proc. ICDCSW*, pp. 470–475, 2004.
16. M. Yiu, Y. Tao, and N. Mamoulis. The B^{dual} -Tree: Indexing Moving Objects by Space-Filling Curves in the Dual Space. *To appear in VLDB Journal*, 2006.

MIME: A Dynamic Index Scheme for Multi-dimensional Query in Mobile P2P Networks

Ping Wang, Lidan Shou, Gang Chen, and Jinxiang Dong

College of Computer Science, Zhejiang University, China

wangping_zju@163.com, {should, cg, djx}@cs.zju.edu.cn

Abstract. Nowadays, as the mobile services become widely used, there is a strong demand for mobile support in P2P search techniques. In this paper, we introduce a new cost model for searching multi-dimensional data in mobile P2P environment and propose a novel multi-dimensional MP2P search scheme MIME. MIME models the physical node layout in a two-dimensional plane and keeps records of the locations of the nodes to construct a proximity-aware P2P overlay. MIME also incorporates two adaptive features: an update algorithm that makes dynamic updates to the overlay, and a cache mechanism that reduces the load of data migration during the updates. Experiment results show that MIME achieves significant performance improvements in point/range queries compared to the conventional system.

1 Introduction

Multi-dimensional search in P2P networks has attracted upon intensive research in the past years, due to the booming of P2P applications. Nowadays, as the mobile services become more widely used, there is a strong demand for mobile support in P2P search techniques. Compared to the conventional P2P network, the Mobile P2P (MP2P) network provides a more constrained communication environment, which is characterized by much more limited bandwidth, higher rate of transmission errors, and the probability that established routes become broken due to mobility. Thus, a physically long route path, which is regarded as inefficient in a conventional P2P network, might be unacceptable in a MP2P case. Based on this observation, we believe that the cost model of the conventional P2P search systems cannot adequately represent search cost in the mobile environment. Therefore, for the MP2P network we advocate a new communication cost model in which the cost of a query resolution is measured in terms of the physical length of its network layer path.

Based on the adoption of the new cost model, we see two technical problems that existing P2P search systems cannot properly address. First, the conventional P2P search algorithms do not take the physical topology of the network into account, resulting in their data allocation and overlay organization schemes not optimized regarding to the network structure. Though the conventional P2P search systems adopt classic spatial database approaches as data allocation scheme to pursue data partition locality, the unawareness of the nodes' physical layout might end up in a situation

where the closely-overlaid nodes are physically far separated and the resolution of a complex query still requires a long physical route. Furthermore, we believe that the efficiency of the conventional P2P overlays needs to be re-examined, as an overlay hop can be implemented with length variable physical paths. Second, in the mobile environment, the performance of a static P2P search system degrades even it is initially efficient, as the motion of the nodes would cause the data allocation and the overlay organization schemes to mismatch the physical network to a greater extent.

To efficiently support multi-dimensional queries in MP2P networks, we propose a novel scheme called Multi-dimensional Index in Mobile Environment (MIME) that addresses the above two problems. MIME captures the physical network layout in a two-dimensional plane and keeps records of the physical locations of the nodes to construct a proximity-aware P2P overlay. MIME also incorporates two adaptive features: an update algorithm that makes dynamic updates to the overlay, and a cache mechanism that reduces the load of data migration during the updates.

The remainder of the paper is organized as follows. Section 2 discusses related work. In Section 3, we present the new query cost model and the novel index technique in MIME. In Section 4, we propose an update algorithm and a cache mechanism. Section 5 presents and analyzes the experimental results. Finally, Section 6 concludes the paper and describes possible future work.

2 Related Work`

Data Allocation Scheme: The data allocation schemes of existing P2P systems can be categorized into two types, namely *DHT-based schemes* and *locality-preserving schemes*. DHT-based schemes [8, 12] differ from locality-preserving schemes in the way data are distributed to nodes and in the search functionalities they provide. The DHT-based schemes use consistent hashing to implement uniform data allocation and only allow data lookup using unique data identifiers. The locality-preserving schemes are adapted from spatial index techniques (e.g. kd-tree [2], Hilbert-curve [6], Z-curve [7]) and often deployed in multi-dimensional P2P search systems (MURK [5], SCRAPER [5], Squid [10], SkipIndex [16]) for supporting complex queries.

Communication Cost Model: Recently, the P2P research community has seen a few works focusing on reducing the discrepancy between the virtual overlay and the underlying physical network. The resulting overlay structures, which are aware of the physical network topology, are usually referred to as proximity-aware or topology-aware routing facilities. For example, Ratnasamy et al. [9] propose a node binning strategy to construct a topology-aware CAN overlay; in [4], Castro et al. compare three existing topology-aware routing approaches, namely proximity routing, topology-based nodeID assignment and proximity neighbor selection; Zahn et al. [14, 15] take a further step to propose an architecture that creates proximity-aware overlays in the mobile environment based on the Random Land Marking (RLM) method. These works also introduce alternative cost metrics such as network latency and the number of network layer hops to replace the number of overlay hops.

Overlay Structure: A number of P2P overlay structures have been deployed to support distributed multi-dimensional search, for example, CAN [8], Chord [12], and Skip Graphs [1]. Shu [11] compares the route performances and maintenance costs of these structures through intensive simulation. In her results, Skip Graph achieves more efficient routing than other overlay structures given the same maintenance cost. However, as CAN has already adopted the notion of a d-dimensional Cartesian coordinate space, it is less demanding to introduce a physical space into CAN.

3 Proximity-Aware Search Method of MIME

3.1 Query Cost Model

In our new cost model, the query resolve cost is given by

$$Q = \sum_{AB \in \text{overlay_route}(\text{query})} C(n_A, n_B) . \quad (1)$$

where $\text{overlay_route}(\text{query})$ indicates the set of all overlay hops in the resolution route of the query. The communication cost of an overlay hop is given by

$$C(n_A, n_B) = \sum_{\text{hop}_i \in \text{path}(n_A, n_B)} f_i(\text{length}(\text{hop}_i)) . \quad (2)$$

where function $f_i(x)$ is usually a linear function representing the cost of the i th network layer hop of the overlay hop. We can design different $f_i(x)$ to obtain communication cost models for different communication environments. In our preliminary experiment, we use the simplified function $f_i(x) = x$, i.e. $C(n_A, n_B)$ is calculated by summing up $\text{length}(\text{hop}_i)$. As we replace the simple counting of overlay hops with a more accurate function, the new cost model is able to capture the more complicated communication condition for the mobile environment.

3.2 Proximity-Aware Index Scheme

Similar to existing multi-dimensional P2P search systems, MIME builds a distributed index for multi-dimensional search based on two components. One is a data allocation method that partitions the multi-dimensional data space into so-called “zones” and maps the zones to the computer nodes; the other is an overlay organization scheme that interconnects the computer nodes for forwarding query messages to relevant nodes. What MIME differs from the conventional P2P search systems is that it introduces the conception of a physical space. That is, MIME models the physical network layout in a two-dimensional physical space and assumes that each node knows its physical/geographic location. Then MIME constructs the proximity-aware index during the partition process of its two spaces (a data space and a physical space), which is triggered by the arrival of a new node as following. First, the new node informs its physical

location to a bootstrap node, which then issues a query to find the new node's carrier node. By carrier node, we mean the node currently maintains the physical zone where the new node resides. Second, the carrier node splits its physical zone into two parts and hands over to the new node the child physical zone in which the new node resides. The dimensions of the physical space are used cyclically for splitting, and the split position is the middle point of the physical locations of the two nodes along the split dimension. Each physical zone is assigned a bit string as its address. The initial zone, i.e. the entire physical space, has an address of null. The addresses of other physical zones are generated by extending those of their parents, i.e. the address of a left/right child zone (which resides lower/upper in the splitting dimension) is generated by appending 0/1 to that of its parent. Third, the carrier node conducts the data space split that is similar to the above physical space split. Note that in MIME the physical locations of the two nodes also determines which data zone is assigned to which nodes. If the new node lies in a position lower than the carrier node along the splitting dimension, it obtains the left physical zone and the left data zone; otherwise, the new node gets the right physical and data zone. Under this assignment policy, the physical zone and the data zone of a node always have the same address.

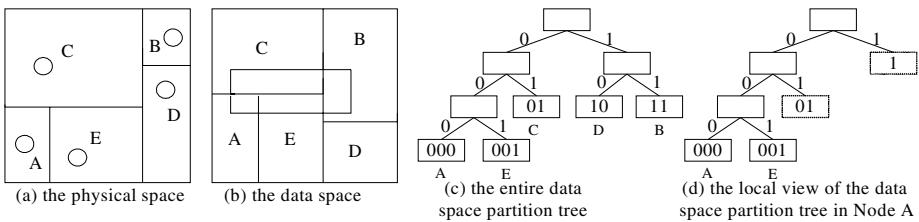


Fig. 1. A dual-space partition and the resulting kd-tree structures

Dual-Space Organization: The physical zones resulting from the physical space split are connected into a CAN overlay, i.e. each node maintains links to its neighbor nodes, i.e. those nodes whose physical zones adjoin its own physical zone. In order to support multi-dimensional queries, these overlay links contain the current ranges of the data zones of the neighbor nodes.

The resulting data zones are organized according to a distributed kd-tree structure, in which each leaf represents a data zone being maintained by a node and each node keeps a partial view of the entire kd-tree. The partial view of the kd-tree in each node is maintained in a data structure similar to the split history proposed in [16], which stores the path from the root of the kd-tree to the respective leaf. Specifically, it consists of a list of tuples in the form of <split dimension, split position> with each entry recording along which dimension and at which position the split occurs. Fig. 1 gives an example dual-space, including the current partition of the physical/data space and the resulting kd-tree structures.

Discussion: The advantage of the above dual-space partition algorithm is that it generates an overlay that is consistent with the physical node layout, i.e. a proximity-aware overlay. We illustrate how a proximity-aware overlay achieves lower communication

cost than a proximity-unaware one with the example in Fig. 2. The proximity-unaware CAN overlay in Fig. 2.a is resulted from the conventional space partition algorithm, the proximity-aware CAN overlay in Fig. 2.c is generated by MIME's partition algorithm, and the plane in Fig. 2.b is the common physical space of the overlays. In Fig. 2.a there are two nodes whose overlay locations are inappropriate, namely B and E. Such kind of mismatches between the overlay locations and the physical locations of the nodes would possibly cause unnecessary communication overhead, e.g. an overlay route from A to E would have to detour to B (route 1). In contrast, in the proximity-aware overlay this overlay route is implemented with a direct network hop from A to E (route 2).

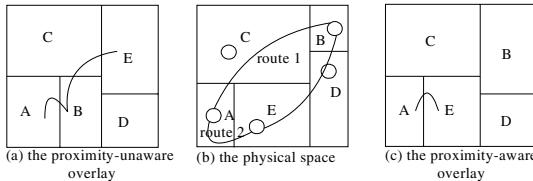


Fig. 2. Proximity-Unaware Overlay vs. Proximity-Aware Overlay

3.3 Point/Range Query Algorithms

MIME resolves a point query in a greedy forwarding manner: each time a node receives a point query message, it examines the data zone ranges of its neighbor nodes and chooses the neighbor node whose data zone is closest to the point being queried as the next hop. This forwarding process repeats until the destination node is reached.

Upon receiving a range query, a MIME node first checks if its data zone intersects the range being queried (noted as QRange). If the check returns false, it forwards the range query to the neighbor node whose data zone is closest to Qrange; if the check returns true, it performs a local search and then traverses its local view of the kd-tree to find other zones that intersect QRange, which are referred to as remote zones. Since a node's local view of the kd-tree is stored in its split history, the kd-tree traverse is implemented by comparing QRange with the split records of the split history. When processing a splitting record, there are three possible cases: (1) If QRange lies in the same halve as the node's data zone, with regard to the split position along the split dimension, we continue to process the next split record to get a longer zone address. (2) When QRange lies in a different halve from the node's data zone, which means that the split history contains no more information about QRange as it branches out of the node's splitting path, a target zone is identified. (3) QRange crosses the split position and covers both of the halves, we obtain a target zone meanwhile we also continue to process the next split record. For example, in Fig. 1 Node A decomposes the range query represented by the rectangle into Z1, Z01, and Z001.

There are two different types of remote zones for a node: the *definite* zones which are managed by the node's neighbors and the *obscured* zones, which the node does not have enough knowledge about whether they have been further partitioned. The

existence of obscured zones is due to the distributed nature of the kd-tree: each node only has a partial view of the kd-tree rather than the entire one. However, the uncertainty of the type of a remote zone will not affect the routing of a range query. For each identified remote zone, denoted as Z , we always choose the neighbor node whose data zone is closest to Z 's centroid as the next hop, no matter whether Z is a definite zone or an obscured one. The next hop node, N_{next} , upon receiving the query, will decompose Z and deliver the range query to the newly generated remote zones if its own local view of the kd-tree contains more information about the partitioning of Z . The pseudo code of this step-by-step refined range query algorithm can be found in the extended version of this paper [13].

4 Providing Mobile Support in MIME

4.1 Update Algorithm of MIME

The update algorithm of MIME runs periodically to check the physical distances that the nodes have moved during the last session and force the excessively moved nodes to rejoin the system using its latest locations. The update algorithm uses a threshold to decide whether a node has moved too far away.

The rejoin process consists of two steps: the *leave* step and the *join* step. Before a node leaves MIME, it needs to find a node to take charge of its physical and data zone. If the leaving node can find its sibling, the other child generated by the splitting of its parent node, the two nodes are merged to reform the parent node; otherwise, the leaving node picks the physically closest neighbor to temporarily charge its physical and data zone. This temporary charging will be released once a new node, located in the leaving node's physical zone, joins the system. The join process is the same as the one we discussed in section 3.2, except that the physical location of the node is obtained at the latest moment.

4.2 Cache Mechanism of MIME

To reduce the load of data migration during the updates to the overlay, MIME incorporates a cache mechanism. A MIME node caches data on two occasions. First, when a node leaves its previous location, it stores the data points of its own data zone into its cache. Thus, when a node moves to a location that it has visited before, it can get the data points from its cache if they have not been replaced. Second, during the joining of a new node, the splitting node will break its data zone into two parts and store the child zone belonging to the new node into its cache. Therefore, if the new node leaves shortly, there is no need to transfer these data back to the splitting node, which has kept them in its cache. The cache mechanism of MIME differs from other caches in two points: First, it requires nodes to cache data in the unit of a data zone. Second, the main purpose of deploying a cache in MIME is to reduce the bandwidth cost in the network, rather than to cut down latency and to increase data accessibility.

5 Experimental Results

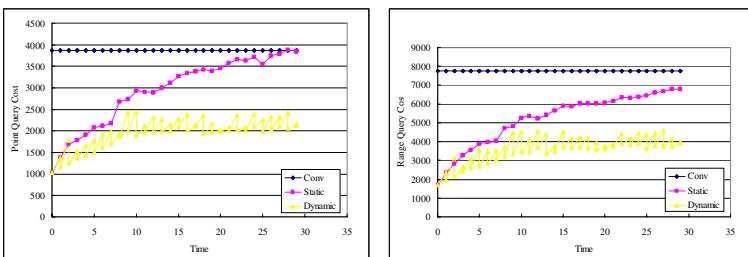
5.1 Experiment Environment

To evaluate and analyze the performance of MIME, we develop a simulator for MIME. We use synthetic datasets for the experiments: the data space is a 1000*1000 two dimensional plane, where up to 100000 data points are uniformly distributed; the physical space is a square region of 1000m*1000m where up to 1000 nodes located. Our sampling interval is 120 seconds. For each experiment, we collect the samples in a whole period of one hour, which contains 30 samples. At the beginning of the sampling, the nodes are uniformly located in the physical space. During the sampling, the nodes move according to either of the following two moving patterns: (1), all nodes move around following the Random Waypoint Model (RWP) [3], with a constant speed ranging from 0.5m/s to 10m/s, and a pause duration of 20 seconds; (2), from 5%-25% of the nodes moves along its own cyclical route, which is on the boundary of a 200m *200m square. We refer to the first moving pattern as RWP and the second moving pattern as CYC.

Each time we randomly select 10 nodes with each node issues 100 random point/range queries. The range queries use squares of 100*100 in the data space. The metrics used for evaluating the performance of the search algorithms are: the average point query cost and the average range query cost. In the following experiments, we compare the search performance of a dynamic (i.e. update-on) MIME, a static (i.e. update-off) MIME and a conventional proximity-unaware system (referred to as CONV). CONV uses the kd-tree as the data allocation scheme and CAN as the overlay structure. Our default parameter setting is: 200 nodes moving under RWP with the speed set to 0.5m/s.

5.2 Performance Improvements of MIME

Fig. 3 compares the point/range query cost of Dynamic MIME, Static MIME, and CONV under the default parameter setting. As showed in this figure, CONV appears to be the upper bound and Static MIME is gradually approaching CONV as time goes on. In contrast, the update algorithm enables Dynamic MIME to remain at a considerably lower point/range query cost, which is about half of CONV's query cost.



(a) Point Query Performance Comparison (b) Range Query Performance Comparison

Fig. 3. Effect of the Update Algorithm

5.3 Tuning the MIME Simulator

(1) Speed of the nodes. Table 1 presents the results of tuning the speed of the nodes. In the table, column “PQ IMP” and “RQ IMP” stand for the average point/range query cost improvement for each update (in percentage); “RJ Nodes” shows the average percentage of the rejoined nodes; “DMNC”/ “DMC” shows the average percentage of the migrated data with the cache turned off/on.

From the first four rows in the table, we can see that RJ Nodes and DMNC/DMC increase as the speed increases, while the query improvements contributed by each update remain almost constant, about 11%. The last two rows indicate a tradeoff between query cost improvement and data migration: if we keep RJ Nodes less than 10% and data migration less than 15%, the query improvements decrease significantly. The last two columns compare the amount of data migration when the cache is turned off and on. The cache size is 2 zones and the average number of data points per cached zone is 22. We can see that under RWP the cache only reduces the amount of data migration marginally. This is expected, as in the RWP model, the possibility that a node revisits a location is small.

Table 1. The Query Improvements and the Amount of Data Migration after Each Update

Speed	PQ IMP	RQ IMP	RJ Nodes	DMNC	DMC
0.5	0.1175	0.1460	0.0552	0.0963	0.0758
1	0.1101	0.1124	0.0790	0.1261	0.1023
5	0.1037	0.1033	0.1622	0.1634	0.1371
10	0.1117	0.1119	0.1755	0.1953	0.1658
5'	0.0395	0.0367	0.0872	0.1172	0.1020
10'	0.0486	0.0433	0.0981	0.1278	0.1074

(2) Effect of the cache. Fig. 4 presents the average percentage of the migrated data when the nodes are moving under CYC. It appears that when the cache is on, even when a large percentage of the nodes move (25%), the migrated data is still limited. Hence, the cache mechanism is especially effective in saving data migration in the occasions that nodes tend to visit a location repetitively. The cache capacity here is 4 zones for each node, and the average number of data points per cached zone is 11.

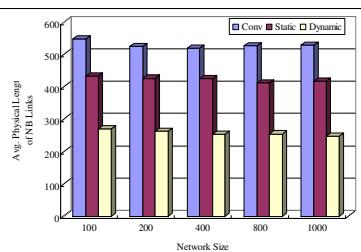
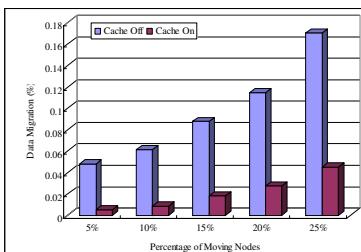


Fig. 4. Data Migration Under CYC

Fig. 5. Average Physical Length of NB Links

(3) Node number of MIME. Fig. 5 plots the average physical length of the neighbor links in Dynamic MIME, Static MIME and CONV at various network sizes (number of nodes). The figure indicates that the average physical length of the neighbor links in Dynamic MIME is about half of that in CONV. These shorter neighbor links enable Dynamic MIME to resolve queries more efficiently than other systems.

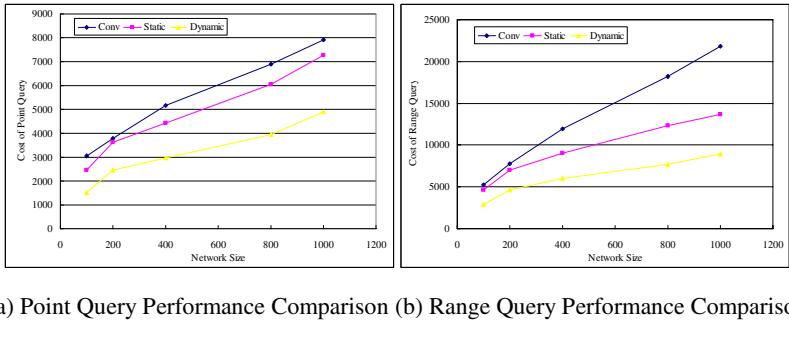


Fig. 6. Query Performance Comparison at Different Network Sizes

Fig. 6 depicts the query performances of Dynamic MIME, Static MIME and CONV at different network sizes. As Fig. 6 shows, the point/rang query cost of Dynamic MIME is significantly lower than that of the other two systems. Fig. 6 also reveals that Dynamic MIME has better scalability than Static MIME and CONV.

6 Conclusions and Future Work

In this paper, we introduced a new cost model for searching multi-dimensional data in MP2P networks. Based on this cost model, we proposed a novel proximity-aware index scheme MIME, which incorporates two adaptive features: an update algorithm that makes dynamic updates to the overlay, and a cache mechanism that reduces the load of data migration during the updates. Simulation results suggested that MIME achieves significant performance improvements compared to the conventional system.

Our future work includes further implementation and evaluation of the search system with more realistic moving patterns and data sets. We also plan to investigate the problem of load balancing under skewed/dynamic data sets.

Acknowledgements. The project was supported in part by the National Science Foundation of China (NSFC, No. 60603044).

References

- [1] J. Aspnes and G. Shah. Skip graphs. In SODA, 2003.
- [2] J. L. Bentley. Multidimensional binary search trees used for associative searching. In CACM, 1975.

- [3] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. In WCMC, 2002.
- [4] M. Castro, P. Druschel, Y. C. Hu, and A. I. T. Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. In FuDiCo, 2003.,
- [5] P. Ganesan, B. Yang, and H. G. Molina. One torus to rule them all: Multidimensional queries in p2p systems. In WebDB, 2004.
- [6] H. Jagadish. Linear clustering of objects with multiple attributes. In SIGMOD, 1990.
- [7] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In PODS, 1984.
- [8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In SIGCOMM, 2001.
- [9] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In INFOCOM, 2002.
- [10] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In HPDC, 2003.
- [11] Y. F. Shu. Supporting complex queries in P2P networks. PhD thesis, National University of Singapore, 2005.
- [12] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In SIGCOMM, 2001.
- [13] P. Wang, L.D. Shou, G. Chen, J.X. Dong. MIME: A dynamic index scheme for multi-dimensional query in mobile P2P networks (extended version)
<http://db.zju.edu.cn/wiki/index.php/Image:MIME.pdf>.
- [14] R. Winter, T. Zahn, J. Schiller. Random landmarking in mobile, topology-aware peer-to-peer networks. In FTDCS 2004.
- [15] T. Zahn, R. Winter, J. Schiller. Simple, efficient peer-to-peer overlay clustering in mobile, ad hoc networks. In ICON, 2004.
- [16] C. Zhang, A. Krishnamurthy, and R. Y. Wang. Skipindex: Towards a scalable peer-to-peer index service for high dimensional data. In Technical Report, TR-703-04, 2004.

Interval-Focused Similarity Search in Time Series Databases

Johannes Aßfalg, Hans-Peter Kriegel, Peer Kröger, Peter Kunath,
Alexey Pryakhin, and Matthias Renz

Institute for Computer Science, Ludwig-Maximilians Universität München
Ottingenstr. 67, 80538 Munich, Germany
{assfalg,kriegel,kroeger,p,kunath,prakhin,renz}@dbs.ifi.lmu.de
<http://www.dbs.ifi.lmu.de/>

Abstract. Similarity search in time series databases usually deals with comparing entire time series objects or subsequence search. In this paper, we formalize the notion of interval-focused similarity queries which take a set of intervals specifying relevant time frames as additional parameter and compare the time series objects only within this user-defined time focus. We propose an original method to efficiently support interval-focused distance range and k -nearest neighbor queries implementing a filter/refinement architecture. In our broad experimental evaluation we show the superiority of our novel approach compared to existing approaches on several real-world data sets.

1 Introduction

Similarity search in time series databases has attracted a lot of research work recently. Existing work usually focus either on a full comparison, i.e. the entire time series are compared by using an appropriate distance function, or on subsequence matching, i.e. all time series objects that “match” a subsequence are retrieved. However, in many applications, only predefined parts of the time series are relevant for a similarity query rather than the entire time series data. The time intervals of these predefined parts are fixed for all time series. Usually, these parts are specified by the user depending on the analysis focus and change from query to query. We call such type of queries where only a small part of the entire time series is relevant *interval-focused similarity queries*. Obviously, interval-focused similarity is a generalization of a full comparison of the time series. On the other hand, the subsequence matching approach is orthogonal to interval-focused similarity. In interval-focused similarity search, the interval relevant to the query is fixed for all time series objects. In subsequence matching, the matching sequences usually do not correspond to a common time frame.

The notion of interval-focused similarity queries is an important concept in many applications. In stock marketing analysis, the behavior of the courses of different securities is examined w.r.t. a given set of events such as political crises or seasonal phenomena. The time courses need to be compared using interval-focused similarity queries that take only some relevant time periods into account

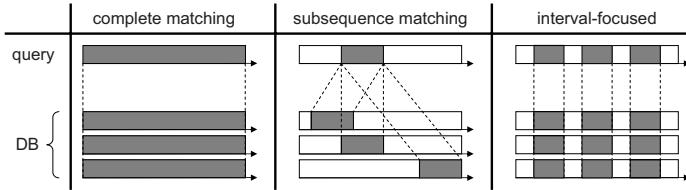


Fig. 1. Different approaches for time series analysis

(e.g. a certain time period after the events). The analysis of the annual balances of a company is usually also focused on specific time intervals (e.g. months), i.e. the balances of specific months are compared using interval-focused similarity queries. In environmental research, the analysis of environmental parameters such as the temperature or the ozon concentration measured over long time periods at various locations usually focus on a given period during the year, e.g. compare the temperatures occurring only in the first week of July each year. Last but not least, in behavior research, brain waves of animals are recorded throughout a given time period, e.g. a day. Researchers often want to compare the brain waves of different individuals during a significant time interval, e.g. during feeding. Obviously, in all these applications, the focus of the analysis task frequently changes from time to time and is not known in advance.

In this paper, we formalize the novel notion of interval-focused similarity queries which is an important generalization of comparing entire time series. In addition, we propose an original method to efficiently support interval-focused distance range and k -nearest neighbor queries that implements a filter/refinement architecture. Furthermore, we discuss how the interval representation approximating the time series can be efficiently accessed using an index structure. The remainder is organized as follows. We discuss related work in Section 2. The novel notion of interval-focused similarity search is formalized in Section 3. In Section 4, we introduce the concept of interval-based representation of the time series. We further show how these representations can be managed efficiently in order to upper and lower bound the distance between time series objects. Based on these bounds we present a filter-refinement architecture to support interval-focused similarity queries efficiently. We discuss two methods for generation interval representations of time series in Section 5. Section 6 provides an experimental evaluation of our proposed methods. Section 7 concludes the paper.

2 Related Work

The (dis)similarity between two time series is usually measured by an appropriate distance function, e.g. the Euclidean distance, Dynamic Time Warping (DTW), Pearson's correlation coefficient, or angular separation, also known as cosine distance. Recent approaches focus either on an entire matching of the query time series with the database objects, or on subsequence matching.

Entire matching approaches consider the complete time course using any of the above mentioned distance measures (cf. Figure 1 (left)). Since the length of a time series is usually very large, the analysis of time series data is limited by the well-known *curse of dimensionality*. The GEMINI method [5] can exploit any dimensionality reduction for time series as long as the distance on the reduced data representation is always a lower bound of the distance on the original data (lower bounding property). In [10], the GEMINI framework is adapted for k -nearest neighbor search. Several dimensionality reduction techniques have been successfully applied to similarity search in time series databases, e.g. [11, 14, 12, 7, 2, 6, 3, 12]. In [9] the authors use a clipped time series representation rather than applying a dimensionality reduction technique. Each time series is represented by a bit string indicating the intervals where the value of the time series is above the mean value of all values of the time series. A distance function that lower bounds the Euclidean distance and DTW is proposed. Obviously, entire matching is a special case of interval-focused similarity. Since all mentioned approximation techniques employing dimensionality reduction or clipping are not designed for interval-focused similarity queries they cannot optimally support this novel query type, especially if the intervals relevant for the query are changing over time and are not known beforehand. In that case, the proposed methods need to approximate the entire time series objects. To answer interval-focused queries these methods need to access the entire approximations rather than only the relevant parts. Subsequence matching approaches usually try to match a query subsequence to subsequences of the database objects (cf. Figure 1 (middle)). The similarity is not affected by the time slot at which o best matches the subsequence q . Usually, a subsequence matching problem is transferred into an entire matching problem by moving a sliding window over each time series object in the database and materializing the corresponding subsequence. If the length of the query subsequence changes, a new sliding window has to be moved over each database time series again. Obviously, subsequence matching is orthogonal to interval-focused similarity. In interval-focused similarity, the time slot relevant for matching is fixed. Two time series are not considered similar even if they have a similar subsequence but at different time intervals. In addition, the concept of interval-focused similarity allows to specify multiple relevant time intervals of different length.

3 Problem Statement and Contributions

Let \mathcal{D} denote a database of n time series. A time series $X = [x_1, \dots, x_N]$ of length N is a sequence of N values, where x_i denotes the value corresponding to the time slot $i \in \mathcal{T} = \{t_1, \dots, t_N\}$ and \mathcal{T} is the domain of time. We assume that all time series are normalized within the interval $[MAX, MIN]$, i.e. $\max_{x_i \in X} x_i = MAX$ and $\min_{x_i \in X} x_i = MIN$ for all time series objects $X \in \mathcal{D}$.

A (time) interval $I = (lT_I, uT_I) \in \mathcal{T} \times \mathcal{T}$ is a pair of time slots where lT_I denotes the start slot and uT_I denotes the end slot. Given a time series $X \in \mathcal{D}$ and an interval I , the *interval sequence of X corresponding to I* is a time series

of length $(uT_I - lT_I) + 1$ consisting of the values of X between the start and the end time slot of I , i.e. $X_I = [x_{lT_I}, \dots, x_{uT_I}]$. A set of k intervals is denoted by $\mathcal{I} = \{I_1, \dots, I_k\}$.

Due to space limitations, we focus on the L_p -norms which are classical distance measures for time series, especially the Euclidean distance ($p = 2$). The proposed concepts can easily be adapted to DTW. The L_p -norm between two time series X and Y is defined as

$$L_p(X, Y) = \sqrt[p]{\sum_{i=1}^N (x_i - y_i)^p}.$$

As discussed above, interval-focused similarity specifies a given part of the time series (i.e. an interval) as relevant, whereas the remaining part of the time series is irrelevant. The relevant part may change from query to query. Let $I = (lT_I, uT_I)$ be a relevant interval. The L_p -norm between X and Y w.r.t. I is defined by

$$L_p^I(X, Y) = \sqrt[p]{\sum_{i=lT_I}^{uT_I} (x_i - y_i)^p}.$$

We want to define interval-focused similarity such that we are not limited to one relevant interval. Rather, we want to be flexible to specify a set of relevant intervals \mathcal{I} that is again specified at query time. Thus, the L_p -norm between X and Y w.r.t. \mathcal{I} is defined by

$$L_p^{\mathcal{I}}(X, Y) = \sqrt[p]{\sum_{I \in \mathcal{I}} L_p^I(X, Y)^p}.$$

Note that the intervals $I \in \mathcal{I}$ can be of varying length and, thus, the influence of each interval on the complete sum may be different. In some applications, it may be interesting to weight the intervals, such that the contribution to the overall distance of each interval is similar. This can be easily achieved by multiplying a weighting factor w_I to each summand. In order to achieve similar influence of each interval I regardless of its length $|I|$, we can set $w_I = 1/|I|$.

Interval-focused distance range query: Given a query time series Q , a distance $\varepsilon \in \mathbb{R}$, and a relevant set of intervals \mathcal{I} , an *interval-focused distance range query* retrieves the set $\text{DRQ}(Q, \varepsilon, \mathcal{I}) = \{X \in \mathcal{D} \mid L_p^{\mathcal{I}}(Q, X) \leq \varepsilon\}$.

Interval-focused k -nearest neighbor query: Given a query time series Q , a number $k \in \mathbb{N}$, and a relevant set of intervals \mathcal{I} , an *interval-focused k -nearest neighbor query* (k NN query) retrieves the set $\text{NNQ}(Q, k, \mathcal{I}) \subseteq \mathcal{D}$ containing at least k time series such that

$$\forall X \in \text{NNQ}(Q, k, \mathcal{I}), \hat{X} \in \mathcal{D} - \text{NNQ}(Q, k, \mathcal{I}) : L_p^{\mathcal{I}}(Q, X) \leq L_p^{\mathcal{I}}(Q, \hat{X}).$$

In this paper, we claim the following contributions: After we have formalized the notion of interval-focused similarity queries, we describe a new efficient representation of time series based on interval boxes in the following. In addition,

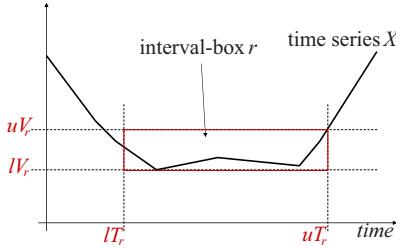


Fig. 2. Illustration of the interval box approximation of a given time series X

we show how this representation can be used to efficiently support interval-focused similarity search using an existing index structure. The key benefit of our novel representation is that we only need to access those parts of the time series objects that are relevant for a given query. Furthermore, we define a lower and upper bound on the interval box representation for any L_p -norm, and describe an efficient multi-step filter/refinement architecture for interval-focused similarity queries.

4 Distance Approximation of Time Series Objects

The basic idea of our approach is to represent each time series object of the database by sequences of intervals. These intervals can be efficiently managed by an index such as the RI-tree [8]. In addition, if we store the maximum and minimum amplitude of the time series within the intervals, these intervals can be used to compute upper and lower bounds of the true distance between different time series. If an interval-focused similarity query is launched specifying a set of relevant time frames \mathcal{I} , only the intervals of the database objects that intersect any $I \in \mathcal{I}$ need to be accessed in order to estimate the lower and upper bounding distance approximations.

4.1 Representing Time Series Objects by Interval Boxes

We approximate each time series $X \in \mathcal{D}$ by a set of intervals. For each interval, we further store the maximum and minimum amplitude of X within the interval. This results in a minimum-bounding box around X within the specified interval (cf. Figure 2) called *interval box*. Formally, an interval box r is given by $r = (lT_r, uT_r, lV_r, uV_r)$, where (lT_r, uT_r) specifies the time interval, $lV_r = \min_{lT_r \leq i \leq uT_r} x_i$, and $uV_r = \max_{lT_r \leq i \leq uT_r} x_i$.

The set of interval boxes approximating X is denoted by $rep(X)$. We discuss methods for generating interval boxes for a given time series later in Section 5. So far, we claim no further constraints for the interval boxes $r \in rep(X)$ as far as $\forall i : lT_r \leq i \leq uT_r : lV_r \leq x_i \leq uV_r$.

4.2 Distance Estimation Using Interval Boxes

In the following, we will discuss how we can estimate the true distance between a query object Q and any $X \in \mathcal{D}$ by means of an upper and a lower bound using the information of $\text{rep}(X)$ rather than using the complete representation of X .

At each relevant time slot i , we can lower bound the i -th summand of the L_p -norm by the well-known *MINDIST* between q_i and any interval box $r \in \text{rep}(X)$ that overlaps i , i.e. $lT_r \leq i \leq uT_r$. The *MINDIST* between q_i and any interval box r with $lT_r \leq i \leq uT_r$ is defined as

$$\text{MINDIST}(q_i, r) = \begin{cases} lV_r - q_i & \text{if } q_i \leq lV_r \\ q_i - uV_r & \text{if } q_i \geq uV_r \\ 0 & \text{else.} \end{cases}$$

If we do not have any interval box $r \in \text{rep}(X)$ that overlaps time slot i , we can only lower bound the true distance between q_i and x_i by 0. If there are several interval boxes $r \in \text{rep}(X)$ with $lT_r \leq i \leq uT_r$, we aggregate the maximum over all the corresponding MINDIST values. Formally, at each time slot i , a lower bound of the i -th summand of the L_p -norm between Q and X is given by

$$LB^i(Q, X) = \max\{0, \max_{\{r \mid r \in \text{rep}(X), lT_r \leq i \leq uT_r\}} \text{MINDIST}(q_i, r)\}.$$

Obviously, $LB^i(Q, X) \leq |q_i - x_i|$. We can now extend the lower bound at each time slot i to intervals $I = (lT_I, uT_I)$ as follows:

$$LB^I(Q, X) = \sqrt[p]{\sum_{i=lT_I}^{uT_I} (LB^i(Q, X))^p}.$$

Still, the lower-bounding property $LB^I(Q, X) \leq L_p^I(Q, X)$ is preserved. A lower bound for a set of intervals $\mathcal{I} = \{I \mid i \in \mathbb{N}^+\}$ is then defined by

$$LB^{\mathcal{I}}(Q, X) = \sqrt[p]{\sum_{I \in \mathcal{I}} (LB^I(Q, X))^p}.$$

Again, we have the lower-bounding property $LB^{\mathcal{I}}(Q, X) \leq L_p^{\mathcal{I}}(Q, X)$.

Analogously, an upper bounding distance estimation can be determined. At each relevant time slot i , we now need to use the *MAXDIST* between q_i and any interval box $r \in \text{rep}(X)$ that overlaps i , i.e. $lT_r \leq i \leq uT_r$, to define an upper bound of the i -th summand of $L_p(Q, X)$. The *MAXDIST* between q_i and any interval box r with $lT_r \leq i \leq uT_r$ is defined as $\text{MAXDIST}(q_i, r) = \max\{|q_i - lV_r|, |q_i - uV_r|\}$.

If we do not have any interval box $r \in \text{rep}(X)$ that overlaps time slot i , we can upper bound the true distance between q_i and x_i by $\max\{|q_i - MAX|, |q_i - MIN|\}$. If there are several interval boxes $r \in \text{rep}(X)$ with $lT_r \leq i \leq uT_r$, we aggregate the minimum over all the MAXDIST values. Formally, at each time slot i , an upper bound of the i -th summand of the L_p -norm between Q and X is given by

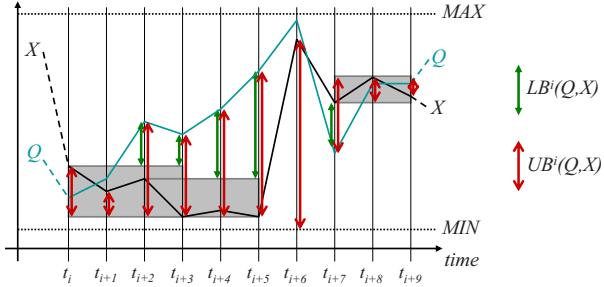


Fig. 3. Lower and upper bounding the L_p -distance within the interval (t_i, t_{i+9})

$$UB^i(Q, X) = \min\{\max\{|q_i - MAX|, |q_i - MIN|\}, \min_{r \in rep(X), lT_r \leq i \leq uT_r} MAXDIST(q_i, r)\}.$$

Analogously, we define

$$UB^I(q_i, x_i) = \sqrt[p]{\sum_{i=lT_I}^{uT_I} (UB^i(Q, X))^p}$$

for time intervals I , and

$$UB^{\mathcal{I}}(Q, X) = \sqrt[p]{\sum_{I \in \mathcal{I}} (UB^I(Q, X))^p}.$$

for sets of time intervals \mathcal{I} . It is easy to prove that $UB^{\mathcal{I}}(Q, X) \geq L_p^{\mathcal{I}}(Q, X)$.

An example for the upper and lower bounding distance estimation is depicted in Figure 3. At time slot t_{i+6} we do not have any interval box representations of X . Thus, the bounds are estimated by $LB^{t_{i+6}}(Q, X) = 0$ and $UB^{t_{i+6}}(Q, X) = \max\{|q_{t_{i+6}} - MAX|, |q_{t_{i+6}} - MIN|\}$. On the other hand, at time slot t_{i+1} the interval box $r = (t_i, t_{i+3}, lV_r, uV_r) \in rep(X)$ is the only interval box that overlaps. We estimate $LB^{t_{i+1}}(Q, X) = MINDIST(q_{t_{i+1}}, r) = 0$ and $UB^{t_{i+1}}(Q, X) = MAXDIST(q_{t_{i+1}}, r) = |q_{t_{i+1}} - lV_r|$.

4.3 Query Processing

In order to compute the upper and lower bounding distance approximations between a query object Q and a database object $X \in \mathcal{D}$ efficiently, we need to determine those interval boxes that intersect the relevant intervals $I \in \mathcal{I}$. For the efficient support of intersection queries, we organize the intervals of the interval boxes in an adoption of the relational interval tree (RI-tree) [8]. An interval intersection query takes a query interval $I \in \mathcal{I}$ and retrieves all intervals in the RI-tree that intersect with I . Details on the processing of intersection

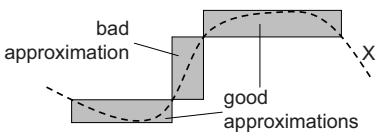
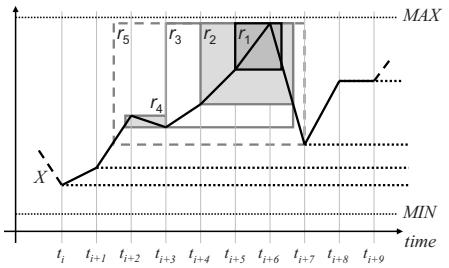
queries using RI-Trees can be found in [8]. In order to determine all interval boxes that intersect with the query intervals we need such an intersection query for all $I \in \mathcal{I}$. This way, we determine for each database object $X \in \mathcal{D}$ those interval boxes $r \in rep(X)$ that intersect with any of the query intervals $I \in \mathcal{I}$ in order to compute $LB^{\mathcal{I}}(Q, X)$ and $UB^{\mathcal{I}}(Q, X)$.

Based on our distance approximations LB and UB introduced above, we can apply the paradigm of filter/refinement query processing to efficiently answer interval-focused distance range and k NN queries. In case of an interval-focused distance range query, we can use both, the upper and the lower bound in the filter step. Each object $X \in \mathcal{D}$ with $LB^{\mathcal{I}}(Q, X) > \varepsilon$ can be identified as true drop because $L_p^{\mathcal{I}}(Q, X) \geq LB^{\mathcal{I}}(Q, X) > \varepsilon$, i.e. $X \notin DRQ(Q, \varepsilon, \mathcal{I})$. On the other hand, each object $X \in \mathcal{D}$ with $UB^{\mathcal{I}}(Q, X) \leq \varepsilon$ can be identified as true hit since $L_p^{\mathcal{I}}(Q, X) \leq UB^{\mathcal{I}}(Q, X) \leq \varepsilon$, i.e. $X \in DRQ(Q, \varepsilon, \mathcal{I})$. In case of an interval-focused k NN query, we can only use the lower bound for the filter step. We apply the approach presented in [10] which is optimal w.r.t. the number of candidates that need to be refined.

5 Generating Approximations

In this section, we will show how to generate adequate interval boxes for a time series. When building the interval boxes we need to take two contradicting considerations into account. On one hand, the number of boxes covering the time series should be low in order to avoid a dramatically increased overhead of the filter step. The performance of the filter step is mainly influenced by the number of interval box approximations to be considered at query time. More boxes lead to higher join cost of the query process. This suggests to construct wide boxes with long intervals. On the other hand, wide boxes will usually worsen the approximation quality since the boxes conservatively approximate the time series. As a consequence, the performance may decrease due to a reduced pruning power of the filter step. This suggests to construct boxes with low approximation error in order to achieve higher values for the lower bounding filter distance $LB^{\mathcal{I}}$ and lower values for the upper bounding filter distance $UB^{\mathcal{I}}$. Following these considerations, the parts of the time series having a flat curvature can be better approximated by interval boxes than parts featuring a high ascending or descending curve (cf. Figure 4 (upper part)). The basic idea of our approach is to optimize the box covering locally. We first identify those parts of the time series which can be well approximated, i.e. subsequences covering the local maximums or minimums of a time series. Then, we try to generate interval boxes that optimally cover the local minimums and maximums of a time series according to a quality criterion given below. Afterwards, we approximate each remaining part of the time series which are not covered yet by one single box.

A high approximation quality of the interval box approximations of a time series is responsible for a good pruning power of our filter step. A high lower bounding distance estimation allows to prune a lot of true drops without the need to refine them. A low upper bounding distance estimation enable to identify some

**Fig. 4.** Interval box approximations**Fig. 5.** Generation of covering boxes

of the true hits without any refinement. For this reason we propose to evaluate the approximation quality of an interval box by considering the expectation of the lower and upper bounding distance between any query object and the approximated part of the database object. For the sake of clarity and due to space limitations, we will focus on the expectation of the lower bound distance w.r.t. an interval box approximation. The expectation of the upper bound distance can be integrated analogously.

Given an interval box $r = (lT_r, uT_r, lV_r, uV_r)$, the expected lower bounding distance $LB^{(lT_r, uT_r)}$ between r and any query time series $Q = [q_1, \dots, q_N]$ which values q_i are assumed to be statistically independent can be computed as follows:

$$E(LB^{(lT_r, uT_r)}(Q, X)) = \sqrt{uT_r - lT_r} \cdot E(LB^i(Q, X)),$$

where

$$E(LB^i(Q, X)) = \int_{MIN}^{MAX} MINDIST(q_i, r) f_i(q_i) = \frac{(MAX - uV_r)^2 + (lV_r - MIN)^2}{2 \cdot (MAX - MIN)}$$

is the expected lower bounding distance according to any time slot $lT_r \leq t_i \leq uT_r$ and $f_i(q_i)$ is the probability density function of the event time series value $q_i \in [MIN, MAX]$. Thereby, we assume that the values of Q are equally distributed between MIN and MAX , i.e.

$$f_i(q_i) = \frac{1}{MAX - MIN}, \forall i \in [MIN, MAX].$$

Now, we can use the expectation of the distance estimations in order to decide for an interval box whether the box setting is more promising than alternative box settings. The higher the expected lower bounding distance w.r.t. an interval box approximation, the higher is its approximation quality.

Next, we will show how interval boxes covering the local extreme values of a time series can be generated nearly optimal according to our quality score. As already mentioned, flat parts, like the local maximums or minimums, of a time series are very adequate for our interval box approximation. We start with the approximation of the local maximums of a time series by searching for each local maximum iteratively in top-down direction. For each local maximum we take all

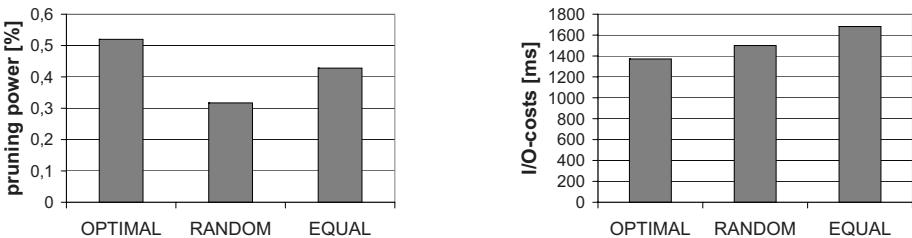


Fig. 6. Evaluation of different interval box generation methods

reasonable conservative coverings into account as shown in the example depicted in Figure 5, and try to pick out the best one. Those interval box candidates which cover or are covered by another interval box candidate are evaluated against each other according to our quality score. The candidate with the highest score is chosen for the approximation, the other candidates will be discarded. This procedure will be applied to all local maximums, so that, finally all local maximums are covered by any interval box. Redundant coverings are removed according to our quality score.

The coverings of the local minimums are generated in the same way. Contrary, this time we start at the local minimums and search the corresponding interval box candidates upwards. After generating all local maximum and minimum coverings, we remove those box candidates which are completely covered by another interval box candidate in order to reduce redundant approximations.

Finally in a post-processing step, the remaining gaps between two adjacent but disjunctive interval boxes, i.e. the parts of the time series which are not covered so far by any interval box, are simply approximated by an additional minimal bounding box. The overall covering of a time series can be performed in $O(N)$ time where N denotes the length of the time series.

6 Evaluation

All experiments were performed on a workstation featuring a 1.8 GHz Opteron CPU and 8GB RAM. We used a disk with a transfer rate of 60 MB/s, a seek time of 3 ms, a latency delay of 2 ms, and a cache allocating 80 KByte. The node capacity of the RI-tree was set to 8 KByte. For each experiment, we launched 100 sample queries and averaged the performance. In each query, we choose the relevant intervals randomly such that the sum of the length of each relevant interval equals the desired query focus size.

We first evaluate our method for generating interval box representations in comparison to two naive solutions on a synthetic data set featuring 400 time series of length 6,000. The first competitor (“RANDOM”) determines a fixed number of intervals randomly and generates a minimum bounding box for each of these intervals. The second competitor (“EQUAL”) works analogously but generates a fixed number of intervals with equal length. The results are shown

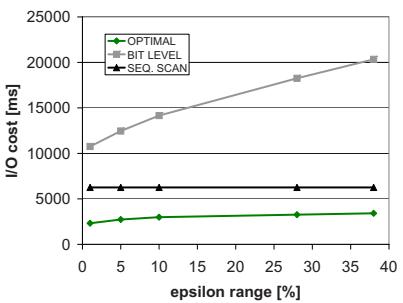
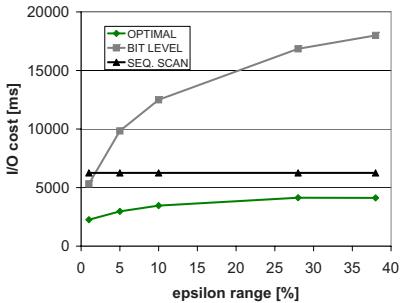


Fig. 7. Performance w.r.t. the selectivity of the query. DS1(left) and DS2 (right).

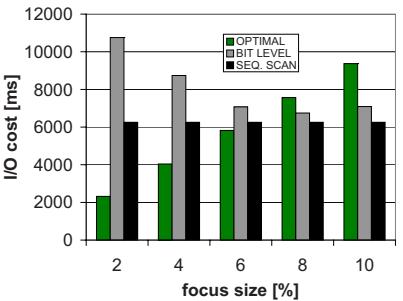
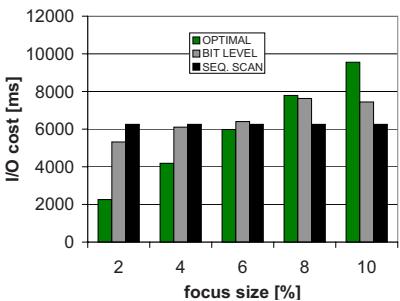


Fig. 8. Performance w.r.t. the size of the query focus. DS1(left) and DS2 (right).

in Figure 6. As it can be seen, our method (“OPTIMAL”) outperforms both competitors in terms of pruning power and I/O cost. This empirically shows that our interval box generation is superior to the two other naive solutions.

Secondly, we evaluate our proposed filter/refinement architecture (OPTIMAL) for answering interval-focused similarity queries compared to the sequential scan (SEQ. SCAN) and the approach proposed in [9] (BIT LEVEL). We choose the second competitor since it is the only approach that does not need to scan the entire time series information for answering interval-focused queries but also proposes a filter/refinement architecture based on a compressed data representation. We used two real-world data sets, “DS1” and “DS2” each featuring 4,800 song-feature time series of length 10,000. The performance of the competitors w.r.t. the selectivity of the query is visualized in Figure 7. The focus size was set to 1% of the time series length. Our approach clearly outperforms both competitors for all settings of the query selectivity. Furthermore, in contrast to “BIT LEVEL” our approach scales well even for large query result sets. The performance of the competitors w.r.t. the size of the query focus is depicted in Figure 8. In this experiment we performed queries featuring a query selectivity of 2% of the dataset. For small focus sizes (< 6% of the time series length) our approach achieved smaller I/O cost than the competing techniques. However, in many applications using

interval focused similarity search a focus size smaller than 5% is reasonable. One can imagine a query on one year records focusing only one certain weak which would correspond to a focus size of about 2%.

7 Conclusions

In this paper, we introduce and formalize the novel concept of interval-focused similarity queries in time series databases which is an important generalization of comparing entire time series. We describe a new efficient representation of time series based on intervals and show how this representation can be used to efficiently support these new query type implementing a filter/refinement approach. Furthermore, we present a method for the generation of the interval-based representation. In our experimental evaluation we show the superiority of our proposed method for answering interval-focused similarity queries in comparison to existing approaches.

References

1. R. Agrawal, C. Faloutsos, and A. Swami. "Efficient Similarity Search in Sequence Databases". In *Proc. 4th Conf. on Foundations of Data Organization and Algorithms*, 1993.
2. O. Alter, P. Brown, and D. Botstein. "Generalized Singular Value Decomposition for Comparative Analysis of Genome-Scale Expression Data Sets of two Different Organisms". *Proc. Natl. Aca. Sci. USA*, 100:3351–3356, 2003.
3. Y. Cai and R. Ng. "Index Spatio-Temporal Trajectories with Chebyshev Polynomials". In *Proc. ACM SIGMOD*, 2004.
4. K. Chan and W. Fu. "Efficient Time Series Matching by Wavelets". In *Proc. IEEE ICDE*, 1999.
5. C. Faloutsos, M. Ranganathan, and Y. Maopoulos. "Fast Subsequence Matching in Time-series Databases". In *Proc. ACM SIGMOD*, 1994.
6. E. Keogh, K. Chakrabati, S. Mehrotra, and M. Pazzani. "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases". In *Proc. ACM SIGMOD*, 2001.
7. F. Korn, H. Jagadish, and C. Faloutsos. "Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences". In *Proc. ACM SIGMOD*, 1997.
8. H.-P. Kriegel, M. Pötke, and T. Seidl. "Interval Sequences: An Object-Relational Approach to Manage Spatial Data". In *Proc. SSTD*, 2001.
9. C. A. Ratanamahatana, E. Keogh, A. J. Bagnall, and S. Lonardi. "A Novel Bit Level Time Series Representation with Implication for Similarity Search and Clustering". In *Proc. PAKDD*, 2005.
10. T. Seidl and Kriegel H.-P. "Optimal Multi-Step k-Nearest Neighbor Search". In *Proc. ACM SIGMOD*, 1998.
11. S. Wichert, K. Fokianos, and K. Strimmer. "Identifying Periodically Expressed Transcripts in Microarray Time Series Data". *Bioinformatics*, 20(1):5–20, 2004.
12. B. K. Yi and C. Faloutsos. "Fast Time Sequence Indexing for Arbitrary Lp Norms". In *Proc. VLDB*, 2000.

Adaptive Distance Measurement for Time Series Databases

Van M. Chhieng and Raymond K. Wong

National ICT Australia & University of New South Wales, 2052 Sydney, Australia
`{vmc,wong}@cse.unsw.edu.au`

Abstract. Efficient retrieval of time series data has gained recent attention from the research community. In particular, finding meaningful distance measurements for various applications is one of the most important issues in the field, since no single distance measurement works for all applications. In this paper, we propose a different distance measurement for time series applications based on Constraint Continuous Editing Distance (CCED) that adjusts the potential energy of each sequence for optimal similarity. Furthermore, we also propose a lower bounding distance for CCED for efficient indexing and fast retrieval, even though CCED does not satisfy triangle inequality.

1 Introduction

In recent years, there are many new meaningful distance measurements [1,7,8,10,12,16,18,20,24] that have been proposed to measure similarity between sequences. In particular, a predominant framework called GEMINI [7] has been adopted by many researchers. GEMINI consists of three phases: insertion, retrieval and filtering. During the insertion phase, high dimensional data is reduced to a lower dimension using Discrete Fourier Transform (DFT), which can then be easily indexed by spatial indexing method such as R^* -Tree [2]. During the retrieval phase, a query is also reduced to a lower dimension using DFT. Then the dimension-reduced query is used to retrieve a temporary result from the database using the R^* -Tree. Finally, during the filtering phase, true distances are computed between the query and data sequences in the temporary result. Sequences whose true distances to the query are more than the specified tolerance are discarded.

Although frameworks such as GEMINI provide a complete setup for querying time series data, the usefulness of such systems rely on the distance measurements employed on the underlying datasets. In particular, previously proposed distance measurements are known to be subjective and their effectiveness are dependent on the properties of the datasets, i.e., noisy/smooth, symmetric/asymmetric, cyclical/non-cyclical and cyclical/non-cyclical. For example, different distance measurements can give different outcomes (i.e., produce different dendrograms) on the same dataset, as shown in the dendrogram diagram [12].

Different from the previously proposed measurements such as Euclidean Distance (L_p -2) [17], Dynamic Time Warping (DTW) [8][10][24], Longest Common Subsequence (LCSS) [20] and its variances (e.g., [14], [19]) that simply processing all the data one by one according to the time axis, this paper presents a distance measurement called Constraint Continuous Editing Distance (CCED) that allows removing, inserting and changing the data values that are abnormal. This difference makes our distance measurement more suitable for applications such as stock data than previous distance measurements such as [8][10][24] since they stretch the time axis for optimal similarity which is semantically incorrect. In addition, CCED constrains the editing path in the cost matrix [17] using global constraints such as Itakura Parallelogram Band [9] and Sakoe-Chiba Band [15] so that the editing path remains close to the main diagonal which gives an intuitive similarity measurement [17]. Furthermore, even if the CCED initially does not satisfy triangle inequality [4], we are able to derive bounds that allow us to index time series data with variances of matrix trees such as M-tree, B-tree or R-tree. Finally, unlike the proposed framework of Chen *et al* [4], our proposed framework has a tuning parameter that allow us to tune for optimal time and space requirements in an adaptive manner. Extensive experiments show that we are able to tune the database for optimal configuration using our approach.

This paper is organized as follows. Section 2 describes the related works. Section 3 discusses Constraint Continuous Editing Distance (CCED). Section 4 presents indexing and retrieval techniques that guarantee no-false dismissal. Section 5 shows the experimental results. Finally, Section 6 concludes the paper.

2 Background

This section consists of two parts. The first part gives an overview of the String Editing Distance (SED) algorithm that is used as a basis for the Constraint Continuous Editing Distance (CCED) algorithm. The second part presents bounding distance to speed up the filtering phase. Table 1 defines conventional notations for the rest of this paper.

Table 1. Conventional Notation

Q	query sequence	$Rest(T)$	all data in T excluding $First(T)$
$\langle \rangle$	empty sequence	D_{SED}	string editing distance
$ T $	length of T	D_{CED}	continuous editing distance
T_i	the i -th element of T	D_{CCED}	constraint continuous editing distance
$First(T)$	the first element of T	D_{lb_CCED}	lower bounding distance of D_{CCED}
p	L_p – norm	β	half global constraint band

2.1 String Editing Distance

String Editing Distance (SED) algorithm was introduced by Levenshtein *et al* [11]. The editing operations permitted by the SED algorithm includes

insert, *delete* and *change* operations. The general structure of SED algorithm for converting sequence Q to T is as follows:

$$\begin{aligned} \mathcal{D}_{SED}(\langle \rangle, \langle \rangle) &= 0 \\ \mathcal{D}_{SED}(T, \langle \rangle) &= |T| \quad // \text{ insert cost} \\ \mathcal{D}_{SED}(\langle \rangle, Q) &= |Q| \quad // \text{ delete cost} \\ \mathcal{D}_{SED}(T, Q) &= \min \left\{ \begin{array}{l} \mathcal{D}_{SED}(T, \text{Rest}(Q)) + 1, \quad // \text{ cost of delete First}(Q) \\ \mathcal{D}_{SED}(\text{Rest}(T), Q) + 1, \quad // \text{ cost of insert First}(Q) \\ \mathcal{D}_{SED}(\text{Rest}(T), \text{Rest}(Q)) + 1, \\ \qquad \qquad \qquad // \text{ cost of change First}(Q) \text{ to First}(T) \end{array} \right. \end{aligned}$$

Given two sequences of discrete values T and Q as shown in Figure 1, a trace sequence [13] describes a sequence of editing operations that transform sequence Q to T . i.e. A line from Q_i to T_j indicates that Q_i is *changed* to T_j , if $Q_i \neq T_j$. This line is also shown from Q_i to T_j , if $Q_i = T_j$. T_i that is not touched by any line corresponds to an *insert* operation on sequence Q . Q_j that is not touched by any line corresponding to a *delete* operation on sequence Q .

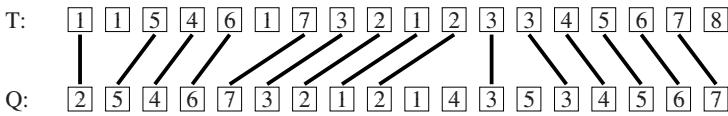


Fig. 1. A trace of operations on a time series sequence of discrete values

The cost of converting a sequence Q to T , or $\mathcal{D}_{SED}(T, Q)$, is the sum of *change*, *insert*, and *delete* operations used as shown in Figure 1. In this particular example, the total cost is 7 since there are seven editing operations. They are as follows: $\{ \text{change}(1,2), \text{insert}(1), \text{insert}(1), \text{delete}(1), \text{delete}(4), \text{delete}(5), \text{insert}(8) \}$.

2.2 Bounding Distances

The time complexity taken to compute $\mathcal{D}_{SED}(T, Q)$ is $O(|T| * |Q|)$. This makes it impractical for applications involving long sequences. In order to reduce the time to compute $\mathcal{D}_{SED}(T, Q)$, bounding distances are normally used.

Property 1. Lower Bounding Distance — $\mathcal{D}_{lb_SED}(T, Q)$

- $\mathcal{D}_{lb_SED}(T, Q) = ||T| - |Q|| \leq \mathcal{D}_{SED}(T, Q)$ or
- $\mathcal{D}_{lb_SED}(T, Q)$ is the number of distinct characters found in Q but not in T .

Property 2. Upper Bounding Distance — $\mathcal{D}_{ub_SED}(T, Q)$

- $\mathcal{D}_{SED}(T, Q) \leq \mathcal{D}_{ub_SED}(T, Q) = \max(|T|, |Q|)$ or
- $\mathcal{D}_{ub_SED}(T, Q)$ is at most the Hamming distance between T and Q , given $|T| = |Q|$.

These bounds, $\mathcal{D}_{lb_SED}(T, Q)$ and $\mathcal{D}_{ub_SED}(T, Q)$, are useful because they can determine either a sequence T from the temporary result is in the final result or not in constant time, i.e., $O(1)$. That is, if $\mathcal{D}_{lb_SED}(T, Q) > \epsilon$, then there is no need to compute $\mathcal{D}_{SED}(T, Q)$ because it is always larger than ϵ . Likewise, if $\mathcal{D}_{ub_SED}(T, Q) < \epsilon$, then there is also no need to compute $\mathcal{D}_{SED}(T, Q)$ because it is always smaller than ϵ . Hence during the filtering phase of the GEMINI framework, it is possible to avoid computing $\mathcal{D}_{SED}(T, Q)$ that has a quadratic time complexity [17], by first computing the upper and lower bounding distances that have constant time complexity.

3 Distance Measurement

In the previous section, we have shown how String Editing Distance (SED) can be used to compare similarity between two sequences of discrete values. The new distance formula for measuring similarity between two sequences of continuous values T and Q has the same structure as String Editing Distance, $\mathcal{D}_{SED}(T, Q)$. However, in order to capture the granularity of the continuous values, we need to make some changes to the general formula of $\mathcal{D}_{SED}(T, Q)$. One can visualize the modifications by imagining that T and Q are sequences of potential energies. Converting a sequence of energies Q to T is the same as *inserting*, *deleting*, and *changing* energies on sequence Q . The following algorithm outlines the formal definition of Continuous Editing Distance, $\mathcal{D}_{CED}(T, Q)$.

$$\mathcal{D}_{CED}(\langle \rangle, \langle \rangle) = 0$$

$$\mathcal{D}_{CED}(T, \langle \rangle) = \sum_{i=0}^{|T|} \text{cost}_p(\text{insert}(T_i))$$

$$\mathcal{D}_{CED}(\langle \rangle, Q) = \sum_{i=0}^{|Q|} \text{cost}_p(\text{delete}(Q_i))$$

$$\mathcal{D}_{CED}(T, Q) = \min \begin{cases} \mathcal{D}_{CED}(T, \text{Rest}(Q)) + \text{cost}_p(\text{delete}(\text{First}(Q))), \\ \mathcal{D}_{CED}(\text{Rest}(T), Q) + \text{cost}_p(\text{insert}(\text{First}(Q))), \\ \mathcal{D}_{CED}(\text{Rest}(T), \text{Rest}(Q)) + \text{cost}_p(\text{change}(\text{First}(T), \text{First}(Q))) \end{cases}$$

where $\text{cost}_p(\cdot)$ is L_p -norm and $\text{change}(\text{First}(T), \text{First}(Q))$ is $|\text{First}(T) - \text{First}(Q)|$.

Given two sequences of continuous values T and Q , Figure 2 shows a trace sequence of editing operations that converts Q into T .

So far, we have described a technique that converts Q to T without constraining the editing path in the cost matrix [17]. This however, does not provide a

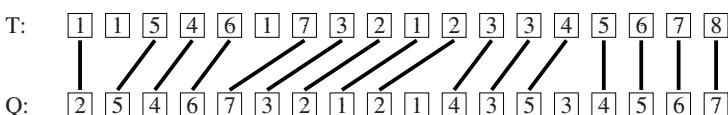


Fig. 2. A trace of operations on a time series sequence of continuous values

meaningful similarity distance as the editing path in the cost matrix can deviate from the main diagonal [3] significantly. In order to constrain the editing path, we use two global constraint techniques: Itakura Parallelogram [9] and Sakoe-Chiba Band [15]. We call CED with global constraints as Constraint Continuous Editing Distance (CCED). In the next three sections, we will discuss the usefulness of CCED, the weakness of previous techniques and the lower bounding distance of CCED.

3.1 Usefulness of CCED

In this paper, it is not our intention to show that our proposed distance measurement is more superior than other knowns distance measurements. Rather, we would like to show the significant differences between our proposed distance measurement compared to others.

The Euclidean distance [17] provides meaningful semantics for distance between points in high dimensions. However, it is also known to be quite brittle when used in similarity search [10]. Dynamic Time Warping (DTW) [8][10][24] is designed for similarity detection in time series data such that values in the data stream are allowed to stretch in the time axis to maximize similarity between sequences. This means that DTW can not be used to measure similarity between time-sensitive data sequence because stretching data points in time axis is semantically incorrect. On the other hand, CCED is different from previous distance measurements. It permits local editing operations such as *insert*, *remove* and *change* within a constraint limit as shown previously.

3.2 Lower Bound of CCED

Using the lower bounding distance property of String Editing Distance (SED) in Section 2.2, the lower bounding distance between two sequences is the difference between the length of the sequence. This is because SED takes into the consideration only discrete values. This lower bounding distance property can be translated into lower bounding distance of Continuous Editing Distance (CED) by replacing length of a sequence by the sum of the potential energies of that sequence. The following formulae illustrates the transition.

$$\mathcal{D}_{lb_SED}(T, Q) = \left| |T| - |Q| \right| \leq \mathcal{D}_{SED}(T, Q) \quad (1)$$

$$\mathcal{D}_{lb_CED}(T, Q) = \left| \sum_1^{|T|} T_i - \sum_1^{|Q|} Q_i \right| \leq \mathcal{D}_{CED}(T, Q) \quad (2)$$

Since the available editing path area in the cost matrix [17] of CCED is less than that in the cost matrix of CED, the distance between two sequences using CCED is always higher than CED. Therefore,

$$\mathcal{D}_{lb_CED}(T, Q) \leq \mathcal{D}_{CED}(T, Q) \leq \mathcal{D}_{CCED}(T, Q) \quad (3)$$

Up to this point, regardless of the value of β (half global constraint band [410]), it is possible to use $\mathcal{D}_{lb_CED}(T, Q)$ as the lower bounding distance for CCED. Hence, it is possible to use various indexing techniques such as B+-tree to index the time series sequence for fast retrieval as has been shown by Chen *et al* [45]. However, there are two problems hidden within this approach. First, B+-tree has only one dimension. This implies that all time series data must be reduced to 1-dimension by summing all potential energies in the sequence. For applications involving long data sequences, such brutal dimensionality reduction technique will over simplify important features of the time series data. Hence, the uses of B+-tree does not have strong discriminating power for similarity search. Second, the lower bounding distance of ERP is not sufficiently tight. As a result this framework has high false positives during the retrieval phase; see Yi *et al* [23] for further detail. In the next section, we introduce a new lower bounding distance that address both issues.

3.3 Tightening Lower Bound for CCED in New Feature Space

Constraint Continuous Editing Distance (CCED) can be viewed as constraining the editing warp path such that the warp path is at most β away from the main diagonal of the cost matrix where β is the constraint function of Sakoe-Chiba or Itakura Parallelogram band. Given this constraint, an envelope of a sequence Q proposed by Keogh *et al* [10] is defined as upper and lower sequences as follows:

$$Q_i^L = \min(Q_{i-\beta} \dots Q_{i+\beta}) \quad Q_i^U = \max(Q_{i-\beta} \dots Q_{i+\beta})$$

Using $\mathcal{D}_{lb_CED}(T, Q)$ shown by Eqn.2 and the envelope of sequence Q , the lower bounding distance of Constraint Continuous Editing Distance, $\mathcal{D}_{lb_CCED}(T, Q)$, is defined as follows:

$$\mathcal{D}_{lb_CCED}(T, Q) = \begin{cases} \sum_{i=1}^{|T|} (T_i)^p - \sum_{i=1}^{|Q|} (Q_i^L)^p & \text{if } \sum_{i=1}^{|T|} (T_i)^p < \sum_{i=1}^{|Q|} (Q_i^L)^p \\ \sum_{i=1}^{|T|} (T_i)^p - \sum_{i=1}^{|Q|} (Q_i^U)^p & \text{if } \sum_{i=1}^{|T|} (T_i)^p > \sum_{i=1}^{|Q|} (Q_i^U)^p \end{cases}$$

Lemma 1. By Yi *et al* [22]: Let $\lambda_1, \lambda_2, \dots, \lambda_\ell$ be non-negative real values such that $\sum_{i=1}^\ell \lambda_i = 1$. If $f(\cdot)$ is convex function and $x \in \mathbb{R}$, then $f(\lambda_1 x_1 + \lambda_2 x_2 + \dots + \lambda_\ell x_\ell) \leq \lambda_1 f(x_1) + \lambda_2 f(x_2) + \dots + \lambda_\ell f(x_\ell)$.

If Q_i^* denotes the envelope of Q , region enclosed by Q_i^L and Q_i^U , and ϵ is the specified tolerance distance (the similarity distance between Q and P), then we have:

$$\begin{aligned} \left| \sum (T_i)^p - \sum (Q_i^*)^p \right| &\leq \epsilon^p && \text{Using the above } \mathcal{D}_{lb_CCED}(T, Q) \\ \sum |(T_i - Q_i^*)|^p &\leq \epsilon^p && \text{Simple reduction} \\ \mathcal{L}_p(T - Q^*) &\leq \epsilon && \text{Using } \mathcal{L}_p \text{'s definition} \\ \ell^{\frac{1}{p}} \sum_i^s \mathcal{L}_p(\vec{F}_i^T - \vec{F}_i^{Q^*}) &\leq \epsilon && \text{Using Lemma 1} \end{aligned}$$

s is the number of segments and
 ℓ is the number of points per segment, [22]

$$\sum_i^s \mathcal{L}_p(\vec{F}_i^T - \vec{F}_i^{Q^*}) \leq \frac{\epsilon}{\ell^{\frac{1}{p}}}$$

□

Furthermore, we have

$$\mathcal{D}_{lb_CCED}(T, Q) = \sum_i^s \mathcal{L}_p(\vec{F}_i^T - \vec{F}_i^{Q^*})$$

This result indicates that if $\mathcal{D}_{lb_CCED}(T, Q)$ is larger than $\frac{\epsilon}{\ell^{1/p}}$, then $\mathcal{D}_{CCED}(T, Q)$ is also larger than ϵ . This property is useful because it can be used during the filtering phase to efficiently discard false positives since the time complexity of $\mathcal{D}_{lb_CCED}(T, Q)$ is $O(s)$ which is much smaller than the time complexity of $\mathcal{D}_{CCED}(T, Q)$ which is $O(n^2)$.

4 Indexing

In order to increase the discriminating power of spatial indexing method, a time series data is first transformed into another vector feature space using segmented means proposed by Yi *et al* [22]. Using their proposal, time series data of length n is reduced to s segments of equal length ℓ . This allows s to be used as a tuning parameter that affects time and space requirement of the framework.

Given a sequence of values denoted as T , \vec{F}_s^T denotes the feature vector of T . Once the time series data has been indexed using R-tree, similarity search can be done using MBR penetration test as shown by Chu *et al* [6]. MBR is a hyper-volume in s dimensions whose hyper-area is defined using container-invariant proposed by Zhu *et al* [24] as follows:

$$L_i = \left[\frac{1}{\ell} \sum_{s(i-1)+1}^{s.i} \text{Min}(Q_i, \beta) \right] - \frac{\epsilon}{\ell^{\frac{1}{p}}}, U_i = \left[\frac{1}{\ell} \sum_{s(i-1)+1}^{s.i} \text{Max}(Q_i, \beta) \right] + \frac{\epsilon}{\ell^{\frac{1}{p}}}$$

$$\text{Min}(Q_i, \beta) = \min(Q_{i-\beta} \dots Q_{i+\beta}), \text{Max}(Q_i, \beta) = \max(Q_{i-\beta} \dots Q_{i+\beta})$$

$$V_s^Q = MBR(Q) = \mathcal{A}_1^s(Q), \mathcal{A}_2^s(Q), \dots, \mathcal{A}_s^s(Q) = \left[\begin{array}{c} U_1 \\ L_1 \end{array} \right], \left[\begin{array}{c} U_2 \\ L_2 \end{array} \right], \dots, \left[\begin{array}{c} U_s \\ L_s \end{array} \right]$$

It is not difficult to see that the penetration test does not produce false dismissal since $\vec{F}_i^{Q^*}$ is $L_i + \epsilon/\ell^{\frac{1}{p}}$ and $U_i - \epsilon/\ell^{\frac{1}{p}}$ — the shaded region shown in Figure 3. The formal proof of no false dismissal is omitted since similar proofs have been provided independently by other researchers [4][10][24].

Using MBR Penetration test, we can select sequences that are within ϵ distance from the query sequence Q using spatial index method such as R-tree. That is, if \vec{F}_s^T does not penetrates all s segments of $\vec{F}_s^{Q^*}$, then $\mathcal{D}_{CCED}(T, Q) > \epsilon$.

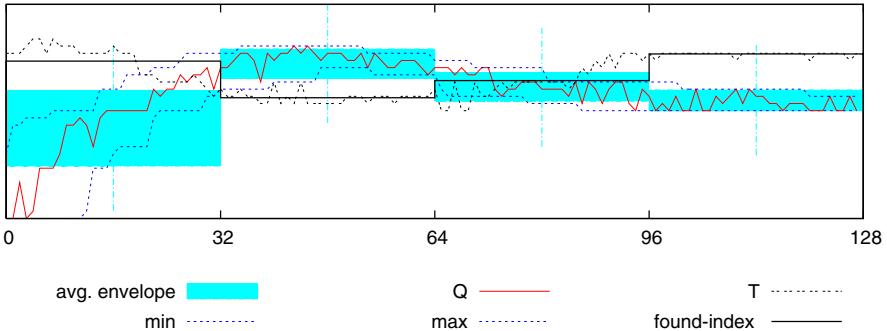


Fig. 3. Decomposition of Hyper-volume of query sequence Q to Hyper-area

After the CCED, the lower bound of CCED and the indexing technique have been described, we are ready to define our approach is as follows:

1. Compute a feature vector space of dimension s for every sequence T .
2. Compute $\bar{F}_s^{Q^*}$ of query Q using the specified properties— β , s , and ϵ .
3. Retrieve sequences that can be within ϵ distance from the query Q .
4. Compute $\mathcal{D}_{lb_CCED}(T, Q)$ to remove false positive from the temporary result.
5. Compute $\mathcal{D}_{CCED}(T, Q)$ for every T in the temporary result for final result.

5 Experiments

In order to verify the effectiveness of our proposed technique, we perform many experiments using time series sequences from real applications. In particular, we study the effectiveness of our technique using random walk data such as stock data since it has been widely used by many researchers [17, 21, 22, 23]. Throughout this section, we undertake extensive experiments that cover every phases of GEMINI framework. Due to the differences in experimental settings, the motivations in the experiments and the lack of common framework, it is not possible to compare our work to Agrawal *et al* [1] since they used Fourier coefficient and R*-tree, Fu *et al* [8, 10] since both main authors focused on Pruning Power for their new findings, and Zhu *et al* [24] since they focused on improving the Pruning Power of Keogh *et al* [10].

Our experimental settings are as follows: CPU=Pentium M 1.80GHz, MEM=512, OS=Ubuntu.dapper, KERNEL=2.6.16, SOURCE=Java.tiger, SPATIAL-INDEX-METHOD=R-Tree, SPATIAL-INDEX-DIMENSION= s , WINDOW=128 points. This section is divided into three parts. Each part describes the performance of each phase of the GEMINI [7] framework: insertion, retrieval and filtering. This enables us to tune the database parameters to achieve the optimal configuration.

5.1 Insertion Phase

During the insertion phase, there are two important aspects we need to consider: the time taken to index time series sequences; and the size of the index. We use two control variables, the number of *data sequences* and the *spatial index dimension*, to access the scalability of both aspects. The results are shown in Figure 4.

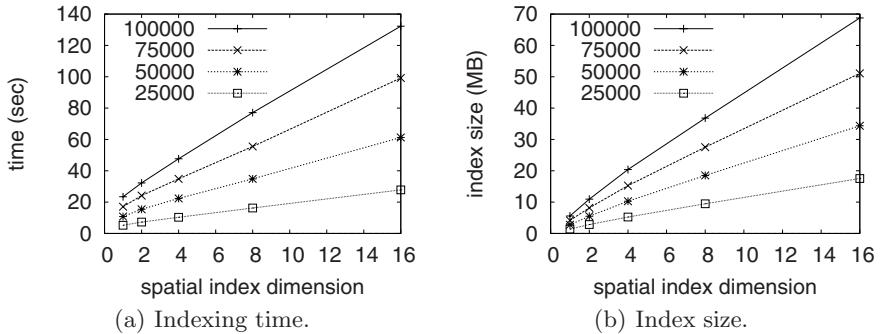


Fig. 4. Insertion Phase

The results shown in Figure 4(a) and 4(b) are as expected, i.e., both time and space taken to index time series data are proportional to both the number of data sequences and the spatial index dimension. This shows that our proposed insertion technique is indeed scalable.

5.2 Retrieval Phase

During the retrieval phase, we perform two experiments regarding the time taken and the number of nodes accessed for retrieving data from the database such that their distances to the query can be within the tolerance distance, ϵ . The time taken and the number of nodes accessed for retrieving data from the database are not influenced by the length of the data sequence, since dimensionality reduction technique is used on the original data. As a result, they can only be influenced by the number of sequences in the database and spatial index dimension. Therefore, we perform experiments based on these two influential variables. The results depicted in Figure 5 show that the choice of spatial index dimension becomes important as the size of the database increases.

In addition to the previous experiments, we perform two more experiments using different variables. The first experiment is based on different max-load per node, whereas in the second experiment, we use query volume as variable. Both experiments show that there is a correlation between the total number of nodes accessed and the number of spatial index dimension as depicted in Figure 6. The results show that the choice of spatial index dimension is important. From our experiment we find that the optimal spatial index dimension is 4 for our dataset.

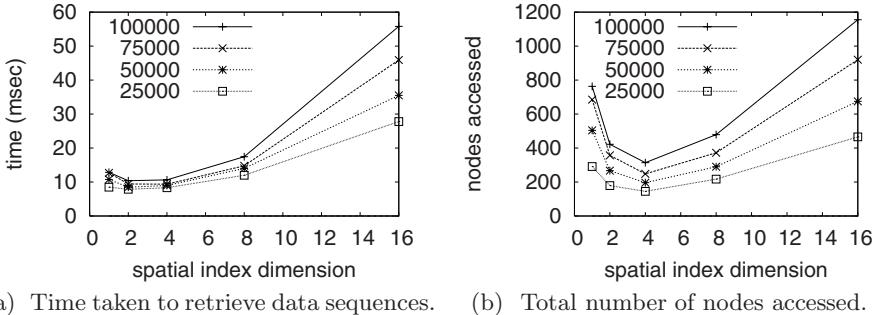


Fig. 5. Retrieval Phase

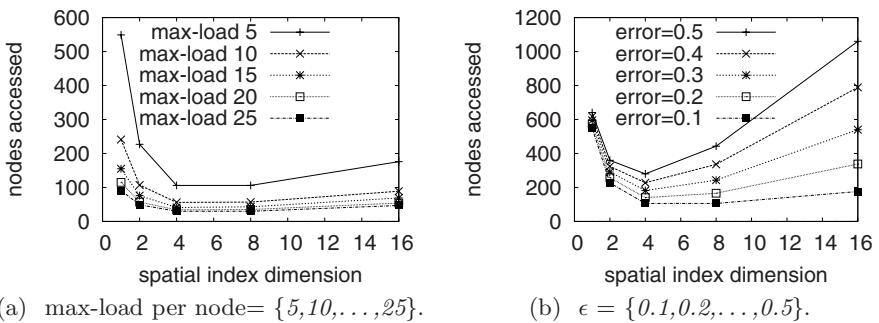
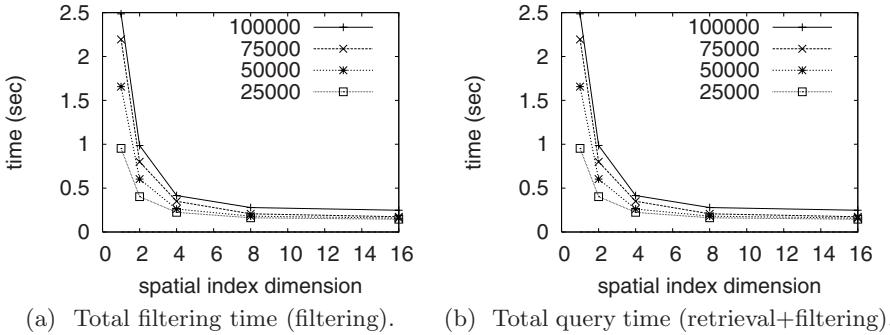
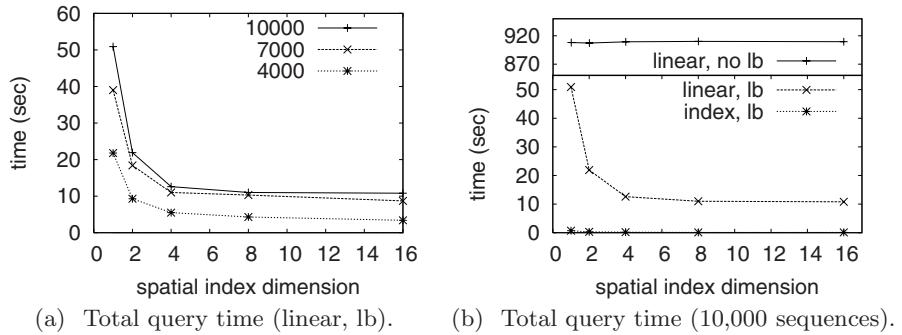


Fig. 6. Retrieval Phase

In order to show that our approach has no false dismissal, we also compare the sequences in the final result produced by spatial index method against those produced by the linear scan approach. The result shows that there is no difference between these two approaches. This is expected as shown in Section 4 that MBR penetration test does not have any false dismissal.

5.3 Filtering Phase

From the results shown in Sections 5.1 and 5.2 high spatial dimension does not always improve the retrieval time. In fact high spatial dimension can degrade the performance of the retrieval phase. This is because the strong discriminating power of high spatial dimension leads to a large number of overlapping hyper-rectangles in similarity search. In order to further study the overall effect of high spatial dimension on the entire framework, we perform two more experiments. In the first experiment, we study the performance of filtering phase using different spatial dimensions, whereas in the second experiment, we study the performance of both retrieval and filtering phases using different spatial index dimensions. The results are shown in Figures 7(a) and 7(b) respectively.

**Fig. 7.** Filtering Phase**Fig. 8.** Filtering Phase

As observed from Figure 7, it is clear that the retrieval phase does not influence the total query time since the total query time is dominated by the filtering phase. However the authors would like to point out that the retrieval phase is indeed useful and effective because it produces small temporary result. To illustrate this further, we perform an experiment that compares the different between linear scan without lower bound, linear scan with lower bound, and index with lower bound. Figure 8(a) shows the performance of the linear scan with lower bounding distance as the size of the database increases. Most importantly, when we compare the performance between all three approaches, our approach (index with lower bound) outperforms all other approaches since our approach produces small temporary result before filtering. The result is shown in Figure 8(b).

6 Conclusion

In order to provide an adaptive distance measurement for various applications that might have different requirement and semantic, this paper has proposed a new distance measurement called CCED that adjusts the potential energy

of each sequence for optimal similarity. In addition, we have developed a lower bounding distance for CCED such that it is possible to efficiently index and query sequences even though CCED does not satisfy triangle inequality. Furthermore, we have proposed a query framework for CCED that consists of two phases: retrieval and filtering. Compared to other work such as Chen *et al* [45], our retrieval method can be set to an optimal configuration (such as optimal spatial index dimension) for any given application. During filtering, CCED computes the similarity between two time series sequences by constraining the editing path in the cost matrix [17] that can lead to more meaningful measurement for different applications. Finally, extensive experiments have shown that our proposed framework is scalable.

References

1. Rakesh Agrawal, Christos Faloutsos, and Arun Narasimha Swami. Efficient similarity search in sequence databases. In *FODO '93: Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, pages 69–84, London, UK, 1993. Springer-Verlag.
2. Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331. ACM Press, 1990.
3. Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. *AAAI-94 Workshop on Knowledge Discovery in Databases*, 1994.
4. Lei Chen and Raymond T. Ng. On the marriage of l_p -norms and edit distance. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *VLDB*, pages 792–803. Morgan Kaufmann, 2004.
5. Lei Chen, M. Tamer Ozsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA, 2005. ACM Press.
6. Kelvin Kam Wing Chu and Man Hon Wong. Fast time-series searching with scaling and shifting. In *PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 237–248, New York, NY, USA, 1999. ACM Press.
7. Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 419–429, New York, NY, USA, 1994. ACM Press.
8. Ada Wai-chee Fu, Eamonn Keogh, Leo Yung Hang Lau, and Chotirat Ann Ratanamahatana. Scaling and time warping in time series querying. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 649–660. VLDB Endowment, 2005.
9. Fumitada Itakura. Minimum prediction residual principle applied to speech recognition. pages 154–158, 1990.
10. Eamonn Keogh. Exact indexing of dynamic time warping. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Data Bases*, pages 406–417, Hong Kong, China, 2002. Morgan Kaufmann.

11. Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics - Doklady*, 10:707–710, 1966.
12. Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11, New York, NY, USA, 2003. ACM Press.
13. Roy Lowrance and Robert A. Wagner. An extension of the string-to-string correction problem. *Journal of the Association for Computing Machinery*, 22(2):177–183, 1975.
14. Saul Needleman and Christian Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal Molecular Biology*, 48:444–453, 1970.
15. Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. pages 159–165, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.
16. Yasushi Sakurai, Masatoshi Yoshikawa, and Christos Faloutsos. Ftw: fast similarity search under the time warping distance. In Chen Li, editor, *PODS*, pages 326–337. ACM, 2005.
17. Stan Salvador and Philip Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. In *KDD Workshop on Mining Temporal and Sequential Data*, 2004.
18. Yutao Shou, Nikos Mamoulis, and David W. Cheung. Fast and exact warping of time series using adaptive segmental approximations. *Mach. Learn.*, 58(2-3):231–267, 2005.
19. Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal Molecular Biology*, 147:195–197, 1981.
20. Michail Vlachos, Dimitrios Gunopoulos, and George Kollios. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684. IEEE Computer Society, 2002.
21. Huanmei Wu, Betty Salzberg, and Donghui Zhang. Online event-driven subsequence matching over financial data streams. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 23–34, New York, NY, USA, 2004. ACM Press.
22. Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 385–394, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
23. Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering*, pages 201–208, Washington, DC, USA, 1998. IEEE Computer Society.
24. Yunyue Zhu and Dennis Shasha. Warping indexes with envelope transforms for query by humming. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *SIGMOD Conference*, pages 181–192. ACM, 2003.

Clustering Moving Objects in Spatial Networks

Jidong Chen^{1,2}, Caifeng Lai^{1,2}, Xiaofeng Meng^{1,2},
Jianliang Xu³, and Haibo Hu³

¹ School of Information, Renmin University of China

² Key Laboratory of Data Engineering and Knowledge Engineering, MOE
{chenjd, laicf, xfmcg}@ruc.edu.cn

³ Department of Computer Science, Hong Kong Baptist University
{xujl, haibo}@comp.hkbu.edu.hk

Abstract. Advances in wireless networks and positioning technologies (e.g., GPS) have enabled new data management applications that monitor moving objects. In such new applications, realtime data analysis such as clustering analysis is becoming one of the most important requirements. In this paper, we present the problem of clustering moving objects in spatial networks and propose a unified framework to address this problem. Due to the innate feature of continuously changing positions of moving objects, the clustering results dynamically change. By exploiting the unique features of road networks, our framework first introduces a notion of cluster block (CB) as the underlying clustering unit. We then divide the clustering process into the continuous maintenance of CBs and periodical construction of clusters with different criteria based on CBs. The algorithms for efficiently maintaining and organizing the CBs to construct clusters are proposed. Extensive experimental results show that our clustering framework achieves high efficiency for clustering moving objects in real road networks.

Keywords: Spatial-Temporal Databases, Moving Objects, Clustering, Spatial Networks.

1 Introduction

Clustering is one of the most important analysis techniques. It groups similar data to provide a summary of data distribution patterns in a dataset. Early research mainly focused on clustering a static dataset [8][11][18][3][13][6][10][4]. In recent years, clustering moving objects has been attracting increasing attention [9][17][7], which has various applications in the domains of weather forecast, traffic jam prediction, animal migration analysis, to name but a few. However, most existing work on clustering moving objects assumed a free movement space and defined the similarity between objects by their Euclidean distance.

In the real world, objects move within spatially constrained networks, e.g., vehicles move on road networks and trains on railway networks. Thus, it is more practical to define the similarity between objects by their network distance – the shortest path distance over the network. However, clustering moving objects

in such networks is more complex than in free movement space. The increasing complexity first comes from the network distance metric. The distance between two arbitrary objects cannot be obtained in constant time, but requires an expensive shortest path computation. Moreover, the clustering results are related to the segments of the network and their changes will be affected by the network constraint. For example, a cluster is likely to move along the road segments and change (i.e., split and merge) at the road junctions due to the objects' diversified spatio-temporal properties (e.g., moving in different directions). It is not efficient to predict their changes only by measuring their compactness. Thus, the existing clustering methods for free movement space cannot be applied to spatial networks efficiently.

On the other hand, the existing clustering algorithms based on the network distance [16] mainly focus on the static objects that lie on spatial networks. To extend to moving objects, we can apply them over the current positions of the objects in the network periodically. However, this approach is prohibitively costly since each time the expensive clustering evaluation starts from scratch. In addition, the clustering algorithms for different clustering criteria (e.g., K -partitioning, distance, and density-based) are totally different in their implementation. This is inefficient for many applications that require to execute multiple clustering algorithms at the same time. For example, in a traffic management application, it is important to monitor those densely populated areas (by density-based clusters) so that traffic control can be applied; but at the same time, there may be a requirement for assigning K police officers to each of the congested areas. In this case, it is favorable to partition the objects into K clusters and keep track of the K -partitioned clusters. Separate evaluation of different types of clusters may incur computational redundancy.

In this paper, we propose a unified framework for “Clustering Moving Objects in spatial Networks” (CMON for short). The goals are to optimize the cost of clustering moving objects and support multiple types of clusters in a single application. The CMON framework divides the clustering process into the continuous maintenance of cluster blocks (CBs) and the periodical construction of clusters with different criteria based on CBs. A CB groups a set of objects on a road segment in close proximity to each other at present and in the near future. In general, a CB satisfies two basic requirements: 1) it is inexpensive to maintain in a spatial network setting; 2) it is able to serve as a building block of different types of application-level clusters. Our contributions are summarized as follows:

- We propose a unified framework for clustering moving objects in spatial networks to efficiently support different clustering criteria at the same time.
- We develop incremental CB maintenance (including split and merge) algorithms by analyzing the object movement features on a spatial network.
- We present efficient algorithms to periodically construct three kinds of clusters based on CBs. The network features are exploited to reduce the search space and avoid unnecessary computation of network distance.
- We show, through extensive experiments, that our clustering algorithms achieve high efficiency.

The rest of the paper is organized as follows. Section 2 surveys the related work. Section 3 describes the proposed framework. Section 4 details the initiation and maintenance of CBs. The algorithms for constructing the clusters with different clustering criteria based on CBs are proposed in Section 5. Section 6 shows experimental evaluations. We conclude this paper in Section 7.

2 Related Work

A lot of clustering techniques have been proposed for static datasets in a Euclidean space. They can be classified into the partitioning [8][11], hierarchical [18][3][13], density-based [10], grid-based [15][1], and model-based [2] clustering methods. There are also a few studies [4][6][16] on clustering nodes or objects in a spatial network. Yiu and Mamoulis [16] defined the problem of clustering objects based on the network distance, which is mostly related to our work. They proposed algorithms for three different clustering paradigms, i.e., *k-medoids* for K -partitioning, ϵ -link for density-based, and *single-link* for hierarchical clustering. These algorithms avoid computing distances between every pair of network nodes by exploiting the properties of the network. However, all these solutions assumed a static dataset. As discussed in the Introduction, a straightforward extension of these algorithms to moving objects by periodical re-evaluation is inefficient. Besides, Jin *et al.* [5] studied the problem of mining distance-based outliers in spatial networks, but it is only a byproduct of clustering.

Clustering analysis on moving objects has recently drawn increasing attentions. Li *et al.* [9] first addressed this problem by proposing a concept of micro moving cluster (MMC), which denotes a group of similar objects both at current time and at near future time. Each MMC maintains a bounding box for the moving objects contained, whose size grows over time. Even the CB in our framework is some kind of micro-cluster, it has much differences from MMC. First MMC is based on the Euclidean distance metric while CB is formed by the network distance. Second, MMC does not consider the network constraint where micro-clusters usually move along the road segment with the objects and change at the road junctions immediately. The prediction of the MMC's split and merge in a spatial network is therefore not accurate. The bounding boxes of MMCs are likely to be exceeded frequently and numbers of maintenance events dominate the overall running time of the algorithms. Finally, as the detailed object information in a MMC is not maintained, it can only support very limited clustering paradigms. While CB uses the distance of neighboring objects to measure the compactness instead of the boundary objects of micro-cluster, it is therefore capable to construct global clusters with different criteria. Afterwards, Zhang and Lin [17] proposed a histogram construction technique based on a clustering paradigm. In [7], Kalogeraki proposed three algorithms to discover moving clusters from historical trajectories of objects. Nehme and Rundensteiner [12] applied the idea of clustering moving objects to optimize the continuous spatio-temporal query execution. The moving cluster is represented by a circle in their algorithms. However, most above works only considered moving objects in

unconstrained environments and defined the similarity between objects by their Euclidean distance. To the best of our knowledge, this is the first work which specifies on the problem of clustering network-constrained moving objects whose similarity is defined by network distance.

3 The System Model and CMON Framework

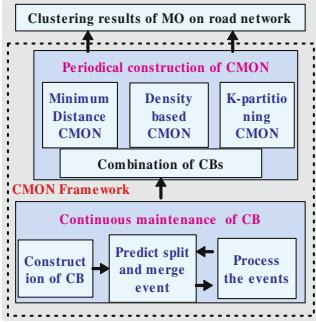
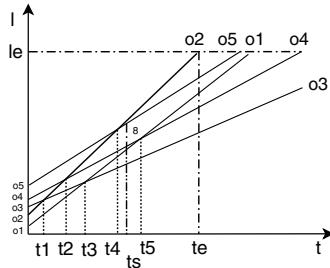
In this section, we describe the system model and present a unified CMON framework for clustering moving objects in a spatial network. It aims to optimize the cost of clustering evaluation and support clusters with different criteria.

We model a spatial network as a graph where objects are moving on the edges [16] (we use the segments interchangeably in this paper). The distance between any two objects, called *network distance*, is measured by the length of the shortest path connecting them in the network. We employ a similar motion model as in [9], where moving objects are assumed to move in a piecewise linear manner (i.e., each object moves at a stable velocity at each edge). We assume that an object location update has the following form $(oid, n_a, n_b, pos, speed, next_node)$, where oid is the id of the moving object, (n_a, n_b) represents the edge on which the object moves (from n_a towards n_b), pos is the relative location to n_a , and $speed$ is the moving speed. We also assume that the next edge to move along, $(n_b, next_node)$, is known in advance. The requirement is to continuously monitor the moving clusters with various criteria at some predefined period.

As shown in Figure 1, the proposed CMON framework is composed of two components: the incremental maintenance of cluster blocks (CBs) and the periodical construction of different types of application-level clusters. A CB is a group of moving objects close to each other at present and near future time. For easy maintenance, we constrain the objects in a CB moving in the same direction and on the same edge segment. Additionally, a CB imposes a strict clustering criterion so as to support different types of application-level clusters. Specifically, the network distance between each pair of neighboring objects in a CB does not exceed a preset threshold ϵ . Formally, a CB is defined as follows:

Definition 1. *Cluster Block.* A cluster block is represented by $CB = (O, n_a, n_b, head, tail, ObjNum)$, where O is a list of objects $\{o_1, o_2, \dots, o_i, \dots, o_n\}$, $o_i = (oid_i, n_a, n_b, pos_i, speed_i, next_node_i)$. Without loss of generality, assuming $pos_1 \leq pos_2 \leq \dots \leq pos_n$, it must satisfy $|pos_{i+1} - pos_i| \leq \epsilon$ ($1 \leq i \leq n-1$). Since all objects are on the same edge (n_a, n_b) , the position of the cluster is determined by an interval $(head, tail)$ in terms of the network distance from n_a . Thus, the length of the CB is $|tail - head|$. $ObjNum$ is the number of objects in the CB.

Note that the edge, position, length, and object number of a CB appear as its summary information. We incrementally maintain each CB by taking into account the objects' anticipated movements. We capture the predicted update events (including split and merge events) of each CB during the continuous movement and process these events accordingly (see Section 4 for details). At any time, clusters of different criteria can be constructed from the CBs, instead

**Fig. 1.** CMON Framework**Fig. 2.** Prediction of Splitting CB

of the entire set of moving objects, which makes the construction processing cost efficient. Moreover, to reduce unnecessary computation of the network distance between the CBs, we adapt the network expansion method to combine CBs to construct the application-level clusters (see Section 5 for details).

4 Maintenance of CBs

Initially, based on the CB definition, a set of CBs are created by traversing all edge segments in the network and their associated objects. The CBs are incrementally maintained after their creation. As time elapses, the distance between adjacent objects in a CB may exceed ϵ and, hence, we need to split the CB. A CB may also merge with adjacent CBs when they are within the distance of ϵ . Thus, for each CB, we predict the time when they may split or merge. The predicted split and merge events are then inserted into an event queue. Afterwards, when the first event in the queue takes place, we process it and update (compute) the split and merge events for affected CBs (new CBs if any). This process is continuously repeated. The key problems are: 1) how to predict split/merge time of a CB, and 2) how to process a split/merge event of a CB.

The split of a CB may occur in two cases. The first is when CB arriving at the end of the segment (i.e., an intersection node of the spatial network). When the moving objects in a CB reach an intersection node, the CB has to be split since they may head in different directions. Obviously, a split time is the time when the first object in the CB arrives at the node. In the second case, the split of a CB is when the distance between some neighboring objects moving on the segment exceeds ϵ . However, it is not easy to predict the split time since the neighborhood of objects changes over time. And therefore the main task is to dynamically maintain the order of objects on the segment. We compute the earliest time instance when two adjacent objects in the CB meet as t_m . We then compare the maximum distance between each pair of adjacent objects with ϵ until t_m . If this distance exceeds ϵ at some time, the process stops and the earliest time exceeding ϵ is recorded as the split time of the CB. Otherwise, we

update the order of objects starting from t_m and repeat the same process until some distance exceeds ϵ or one of the objects arrives at the end of the segment. When the velocity of an object changes over the segment, we need to re-predict the split and merge time of the CB.

Figure 2 shows an example. Given $\epsilon = 7$, we compute the split time as follows. At the initial time t_0 , the CB is formed with a list of objects $< o_1, o_2, o_3, o_4, o_5 >$. We first compute the time t_e when the first object (i.e., o_2) arrives at the end of the segment (i.e., le). For adjacent objects, we find that the earliest meeting time is t_1 at which o_2 and o_3 first meet. We then compare the maximum distance for each pair of adjacent objects during $[t_0, t_1]$ and no pair whose distance exceeds 7. At t_1 , the object list is updated into $< o_1, o_3, o_2, o_4, o_5 >$. In the same way, the next meeting time is at t_2 for o_2 and o_4 . There is also no neighboring objects whose distance exceeds 7 during $[t_1, t_2]$. As the algorithm continues, at t_4 , the object list becomes $< o_3, o_1, o_4, o_5, o_2 >$ and t_5 is the next time for o_1 and o_4 to meet. When comparing neighboring objects during $[t_4, t_5]$, we find the o_4 and o_5 whose distance is longer than 7 at time t_s . Since $t_s < t_e$, we obtain t_s as the split time of the CB.

We now discuss how to handle a split event. If the split event happens on the segment, we can simply split the CB into two ones and predict the split and merge events for each of them. If the split event occurs at the end of the segment, the processing would be more complex. One straightforward method is to handle the departure of the objects individually each time an object reaches the end of the segment. Obviously, the cost of this method is high. To reduce the processing cost, we propose a group split scheme. When the first object leaves the segment, we split the original CB into several new CBs according to objects' directions (which can be implied from *next_node*). On one hand, we compute a *to-be-expired time* (i.e., the time until the departure from the segment) for each object in the original CB and retain the CB until the last object leaves the segment. On the other hand, we attach a *to-be-valid time* (with the same value as *to-be-expired time*) for each object in the new CBs. Only valid objects will be counted in constructing application-level clusters. Figure 3 illustrates this split example. When CB_1 reaches J_1 , objects p_1 and p_3 will move to the segment $< J_1, J_2 >$ while p_2 and p_4 will follow $< J_1, J_6 >$. Thus, CB_1 is split into two such that p_2 and p_4 join CB_3 , and p_1 and p_3 form a new cluster CB_4 . We still keep CB_1 until p_4 leaves $< J_4, J_1 >$. As can be seen, the group split scheme reduces the number of split events and hence the cost of CB maintenance.

The merge of CBs may occur when adjacent CBs in a segment are moving together (i.e. their network distance $\leq \epsilon$). To predict the initial merge time of CBs, we dynamically maintain the boundary objects of each CB and their validity time (the period when they are treated as boundary of the CB), and compare the minimum distances between the boundary objects of two CBs with the threshold ϵ at their validity time. The boundary objects of CBs can be obtained by maintaining the order of objects during computing the split time.

For the example in Figure 2, the boundary objects of the CB are represented by (o_1, o_5) for validity time $[t_0, t_3]$, (o_3, o_5) for $[t_3, t_4]$, and (o_3, o_2) for $[t_4, t_e]$. The processing of the merge event is similar to the split event on the segment. We get the merge event and time from the event queue to merge the CBs into one CB and compute the split time and merge time of the merged CB. Finally, the corresponding affected CBs in the event queue are updated.

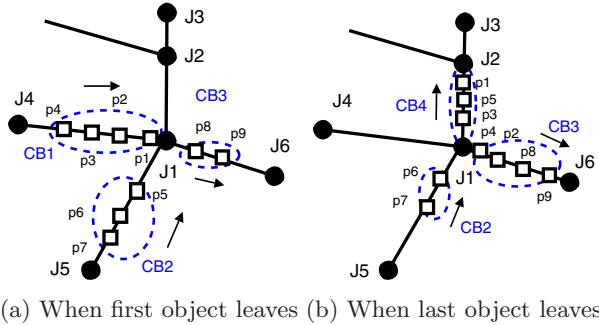


Fig. 3. Group Split at an Edge Intersection

Besides the split and merge of a CB, new objects may come into the network or existing objects may leave. For a new object, we locate all CBs of the same segment that the object enters and see if the new object can join any CB according to the CB definition. If the object can join some CB, the CB's split and merge events are updated. If no such CBs are found, a new CB for the object is created and the merge event is computed. For a leaving object, we update its original CB's split and merge events if necessary.

5 CMON Construction with Different Criteria

This section discusses how to construct application-level clusters of different criteria from CBs. We focus our discussions on three common clustering criteria, i.e., distance-based, density-based, and K -partitioning.

5.1 Distance-Based CMON

A common clustering criterion is based on the minimum distance metric. The Minimum Distance CMON is defined as follows:

Definition 2. *Minimum Distance CMON (MD-CMON).* For each object in an MD-CMON, the minimum network distance with other objects in the cluster is not longer than a user specified threshold δ ($\delta \geq \epsilon$).

The requirement of $\epsilon \leq \delta$ is necessary because it guarantees that a CB does not cross two clusters in the MD-CMON. The MD-CMON can be constructed by

combining the CBs. Generally, for two CBs, we need to compute their network distance (i.e., the minimum network distance of their boundary objects) to determine whether to combine them. This simple method has a time complexity of $O(N^2)$, where N is the number of CBs. In order to reduce the computation cost, we adapt the incremental network expansion method to combine the CBs. The detailed algorithm can be found in Algorithm II.

The algorithm starts with a CB and adds its adjacent nodes that are within δ to a queue Q using Dijkstra's algorithm. Take Figure 4 as an example. Suppose $\delta = 10$ and the algorithm starts with CB_1 . Thus, initially CB_1 is marked "visited" and J_1 is added to Q . The algorithm proceeds to dequeue the first node in Q (i.e., J_1). All adjacent edges of J_1 (except the checked edge $< J_6, J_1 >$) are examined. For each edge $< J_1, J_i >$, assuming $dist(J_1, J_i)$ to be the edge length, if J_i satisfies $dist(CB_1, J_1) + dist(J_1, J_i) \leq \delta$, J_i is added to Q and $dist(CB_1, J_i) = dist(CB_1, J_1) + dist(J_1, J_i)$. Moreover, all unvisited CBs on each adjacent edge are checked. For a CB_i on $< J_1, J_i >$, if $dist(CB_1, J_1) + dist(J_1, CB_i) \leq \delta$, CB_i is merged into CB_1 's MD-CMON cluster. If $dist(CB_i, J_i) \leq \delta$ and J_i has not been added to Q , it is added to Q . The algorithm continues with the same process until Q becomes empty and the CBs around CB_1 are combined into a cluster C_1 . Afterwards, the algorithm picks up another unvisited CB and repeats the same process until all CBs are visited.

5.2 Density-Based CMON

The second clustering criterion is density-based, which is good at filtering out noise data.

Definition 3. *Density-based CMON (DB-CMON).* For each cluster in the DB-CMON, the average density should be higher than a given threshold ρ . Moreover, there should not be any empty segment (without any objects lying on) whose length is longer than E .

Suppose there are $m(m > 1)$ objects in a CB, we have the density of the CB as $\frac{m}{\epsilon(m-1)} > \frac{1}{\epsilon}$. The second condition is necessary to avoid very skewed clusters.

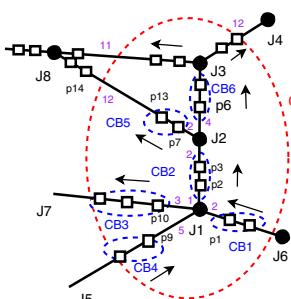


Fig. 4. The Combination of CBs

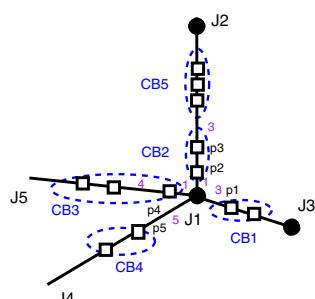


Fig. 5. The Cross-CB

Algorithm 1. MD_CMON()

```

1 foreach  $CB_i$  do
2   if  $CB_i.visited == false$  then
3      $Q = \text{new priority queue};$ 
4     find edge  $n_x, n_y$  where  $CB_i$  lies;
5      $CB = CB_i; C = CB;$ 
6      $nextCB = \text{Next CB on } n_x, n_y \text{ from } CB_i \text{ to } n_y;$ 
7     while ( $nextCB \neq \text{null}$ ) and  $\text{Dist}(CB.\text{head}, nextCB.\text{tail}) \leq \delta$  do
8       Merge_Expand( $CB, nextCB, C, n_x, n_y$ );
9       if ( $nextCB == \text{null}$ ) and  $\text{Dist}(CB.\text{head}, n_y) \leq \delta$  then
10          $B.\text{node} = n_y; B.\text{dist} = \text{Dist}(CB.\text{head}, n_y);$ 
11         Enqueue( $Q, B$ );
12       while  $notempty(Q)$  do
13          $B = \text{Dequeue}(Q);$ 
14         foreach node  $n_z$  adjacent to  $B.\text{node}$  do
15            $nextCB = \text{Next CB from } B.\text{node} \text{ to } n_z;$ 
16           if ( $nextCB \neq \text{null}$ ) and  $\text{Dist}(B.\text{node}, nextCB.\text{tail}) + B.\text{dist} \leq \delta$ 
17             then
18                $newd_{n_z} = \text{Dist}(nextCB.\text{head}, n_z);$ 
19               Merge_Expand( $CB, nextCB, C, B.\text{node}, n_z$ );
20               while ( $nextCB \neq \text{null}$ ) and
21                  $\text{Dist}(CB.\text{head}, nextCB.\text{tail}) \leq \delta$  do
22                    $newd_{n_z} = \text{Dist}(nextCB.\text{head}, n_z);$ 
23                   Merge_Expand( $CB, nextCB, C, B.\text{node}, n_z$ );
24                   if ( $no\ CBs\ on\ edge\ (B.\text{node}, n_z)$ ) then
25                      $newd_{n_z} = B.\text{dist} + \text{Dist}(B.\text{node}, n_z);$ 
26                     if ( $nextCB == \text{null}$ ) and ( $newd_{n_z} \leq \delta$ ) then
27                        $B_{\text{new}}.\text{node} = n_z; B_{\text{new}}.\text{dist} = newd_{n_z};$ 
28                       Enqueue( $Q, B_{\text{new}}$ );

```

Procedure. Merge_Expand($CB_1, CB_2, C, node_1, node_2$)

```

1 if  $CB_2.visited == false$  then
2    $C = \text{MergeClst}(C, CB_2);$ 
3    $CB_1 = CB_2; CB_1.visited = true;$ 
4    $CB_2 = \text{Next CB from } node_1 \text{ to } node_2;$ 
5 else
6    $C_1 = \text{FindCluster}(CB_2);$ 
7    $C = \text{MergeClst}(C, C_1);$ 

```

It is equivalent to the condition that for any object in the cluster, the nearest object is within a distance of E . Thus, to construct the DB-CMON clusters from CBs, we require $\epsilon \leq \max\{E, \frac{1}{\rho}\}$.

The cluster formation algorithm is the same as the one described in Algorithm 1 except that the minimum-distance constraint (transformed from the density constraint) is dynamic. Suppose the density of the current cluster with

k objects is ρ' and a CB has m objects with a length of L . If a CB can be merged into the cluster, their minimum distance D must satisfy $\frac{k+m}{k/\rho'+L+D} \geq \rho$, i.e., $D \leq \frac{k+m+\rho(k/\rho'+L)}{\rho}$.

5.3 K-Partitioning CMON

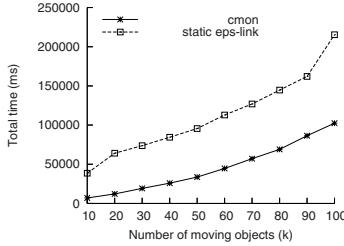
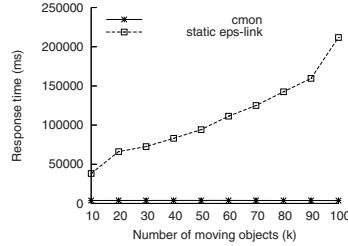
K-Partitioning CMON is similar to the K-Partitioning clustering method [8][11]. It can be defined as follows:

Definition 4. *K-Partitioning CMON (KP-CMON).* Given a set of objects, group them into K clusters such that the sum of distances between all adjacent objects in each cluster is minimized.

According to the definition of CBs, the sum of distances between all adjacent objects in each CB is minimized. Therefore, it is intuitive to construct the KP-CMON clusters from the CBs. An exhaustive method is to iteratively combine the closest pairs of CBs until K clusters are obtained. This method requires to compute the distances between all pairs of CBs, which is costly. Hereby, we propose a low-complexity heuristic similar to the K -means algorithm [8][11]. We initially select K CBs as the seeds for K clusters. For the remaining CBs, we assign them to their nearest clusters to make the sum of distances between adjacent objects to be minimum. Note that this heuristic may not lead to the optimal solution. Suppose that in Figure 5, the distances between CBs are: $dist(CB_2, CB_3) < dist(CB_2, CB_5) < dist(CB_3, CB_1) < dist(CB_2, CB_1) < dist(CB_3, CB_5)$, and that the initial seed CBs are CB_1 and CB_5 for $K = 2$. When CB_3 is checked, it will be assigned to the cluster of $\{CB_1\}$. Then CB_2 will be assigned to the cluster of $\{CB_5\}$, which is different from the optimal solution where CB_2 and CB_3 should be grouped together since $dist(CB_2, CB_3) < dist(CB_2, CB_5)$. To compensate for such mistakes, we introduce the concept of Cross-CB. For adjacent CBs lie around the same node, if their minimum distance is less than ϵ , we group them into a Cross-CB. Then, the clustering algorithm is applied over the CBs and Cross-CBs.

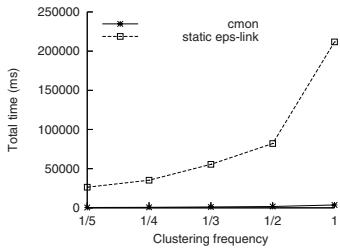
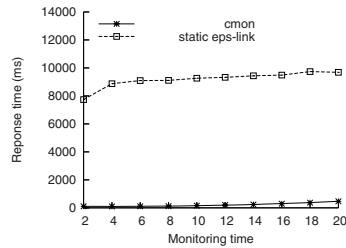
6 Performance Analysis

In this section, we evaluate the performance of our proposed techniques by comparing with the periodical clustering. We implement the CMON algorithms in C++ and carry out experiments on a Pentium 4, 2.4 GHz PC with 512MB RAM running Windows XP. Our performance study uses synthetic datasets. For monitoring the effective clusters in the road network, we design a moving object generator. The generator takes a map of a road network as input, and our experiment is based on the map of Beijing city. We set K hot spots in the map. Initially, the generator places eighty percent objects around the hot spots and twenty percent objects at random positions, and updates their locations at each time unit. Each object around the hot spot moves along its shortest path from its initial hot spot to another one. We compare the MD-CMON algorithm

**Fig. 6.** Total time varies in data size**Fig. 7.** Response time varies in data size

against the ϵ -link algorithm proposed in [16]. We monitor the clustering results by running the ϵ -link algorithm periodically and by maintaining the CBs created at the initial time and combining them to construct the MD-CMON.

First, we compare our method with the static ϵ -link by measuring both average clustering response time and total workload time when varying the number of moving objects from 100K to 1M. We set the clustering frequency at 1 per time unit and execute the CBs maintenance and combination in comparison with the static ϵ -link on all objects. For total workload time (shown in Figure 6), we measure the total CPU time including maintaining CB and combining CBs to clusters up to 20 time units. Figure 7 also shows the average clustering response time for periodic clustering requests. In essence, CBs are like B^+ -tree or R-tree index for periodical queries and they share the same property, i.e., amortizing the query (clustering) cost to maintain the data structure (CBs) for speeding up the query (clustering) processing. Therefore, our method is substantially better than the static one in terms of average response time, yet is still better in terms of total workload time.

**Fig. 8.** Clustering frequency effect**Fig. 9.** Monitoring time effect

Then, we change the clustering frequency with $1/5, 1/4, \dots, 1$ to examine how the total time is affected. The experiment is executed on 100K moving objects during 20 time units. Figure 8 shows the results of the two methods under different clustering frequencies. We can see that the higher the clustering frequency, the more efficient our CMON method. In addition, we fix the clustering frequency at 1 and measure the clustering response time at different clustering monitoring instances. As time elapses, the objects change their locations continuously, which may affect the clustering efficiency. As shown in Figure 9, our

CMON method consistently keeps a lower cost than the static ϵ -link method over different time instances.

Finally, we study the effect of parameters (ϵ and δ) of our methods on the clustering efficiency. As the number of CBs depends on the system parameter ϵ , we change the value of ϵ from 0.5 to 3 to measure the maintenance cost of CBs. Then when fixing the ϵ value at 2.5, we vary δ to study its effect on the CB combination to clusters. Figure 10 and Figure 11 show the effect of the two parameters. We observe that when ϵ and δ are set to 2.5 and 4.5, the method achieves the highest efficiency in our experimental setting.

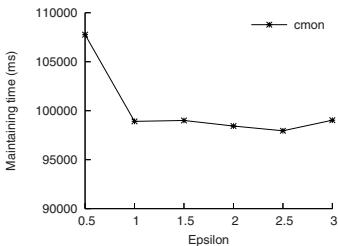


Fig. 10. CMON performance with ϵ

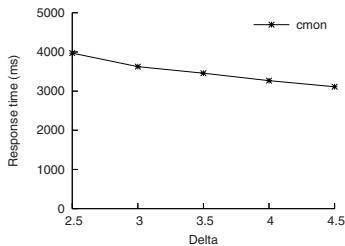


Fig. 11. CMON performance with δ

7 Conclusion

In this paper, we studied the problem of clustering moving objects in a spatial road network and proposed a framework to address this problem. By introducing a notion of cluster block, this framework, on one hand, amortizes the cost of clustering into CB maintenance and combination based on the object movement feature in the road network; and on the other hand, it efficiently supports different clustering criteria. We exploited the features of the road network to predict the split and merge of CBs accurately and efficiently. Three different clustering criteria have been defined and the cluster construction algorithms based on CBs were proposed. The experimental results showed the efficiency of our method.

Acknowledgments

This research was partially supported by the grants from the Natural Science Foundation of China under grant number 60573091, 60273018; Program for New Century Excellent Talents in University (NCET); Program for Creative PhD Thesis in University. Jianliang Xu's work was supported by the grants from the Research Grants Council, Hong Kong SAR, China (Project Nos. HKBU 2115/05E and HKBU 2112/06E).

References

1. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan: Automatic subspace clustering of high dimensional data for data mining applications. SIGMOD 1998: 94-105.
2. D. Fisher: Knowledge acquisition via incremental conceptual clustering. Machine Learning, 1987, 2:139-172.
3. S. Guha, R. Rastogi, and K. Shim: CURE: An efficient clustering algorithm for large databases. SIGMOD 1998: 73-84.
4. A. K. Jain and R. C. Dubes: Algorithms for Clustering Data. Prentice Hall, 1988.
5. W. Jin, Y. Jiang, W. Qian, A. K. H. Tung: Mining Outliers in Spatial Networks. DASFAA 2006: 156-170.
6. G. Karypis, E. H. Han, and V. Kumar: Chameleon: Hierarchical clustering using dynamic modeling. IEEE Computer, 1999, 32(8):68-75.
7. P. Kalnis, N. Mamoulis, S. Bakiras: On Discovering Moving Clusters in Spatio-temporal Data. SSTD 2005: 364-381.
8. L. Kaufman and P. J. Rousseeuw: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley and Sons, Inc, 1990.
9. Y.F. Li, J.W. Han, J. Yang: Clustering Moving Objects. KDD 2004: 617-622.
10. E. Martin, H. P. Kriegel, J. Sander, and X. Xu: A density-based algorithm for discovering clusters in large spatial databases with noise. SIGKDD 1996: 226-231.
11. R. T. Ng and J. Han: Efficient and effective clustering methods for spatial data mining. VLDB 1994: 144-155.
12. R. V. Nehme, E. A. Rundensteiner: SCUBA: Scalable Cluster-Based Algorithm for Evaluating Continuous Spatio-temporal Queries on Moving Objects. EDBT 2006: 1001-1019.
13. A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos: C2P: Clustering based on closest pairs. VLDB 2001: 331-340.
14. D. Papadias, J. Zhang, N. Mamoulis, Y. Tao: Query Processing in Spatial Network Databases. VLDB 2003: 790-801.
15. W. Wang, Yang, R. Muntz, STING: A Statistical Information grid Approach to Spatial Data Mining. VLDB 1997: 186-195.
16. M. L. Yiu, N. Mamoulis: Clustering Objects on a Spatial Network. SIGMOD 2004: 443-454.
17. Q. Zhang, X. Lin: Clustering Moving Objects for Spatio-temporal Selectivity Estimation. ADC 2004: 123-130.
18. T. Zhang, R. Ramakrishnan, and M. Livny: BIRCH:An efficient data clustering method for very large databases. SIGMOD 1996: 103-114.

The Tornado Model: Uncertainty Model for Continuously Changing Data

Byunggu Yu¹, Seon Ho Kim², Shayma Alkobaisi²,
Wan D. Bae², and Thomas Bailey³

¹ Computer Science Department, National University
La Jolla, CA 92037, USA
byu@nun.edu

² Computer Science Department, University of Denver
Denver, CO 80208, USA
{seonkim,salkobai,wbae}@cs.du.edu

³ Computer Science Department, University of Wyoming
Laramie, WY 82071, USA
tbailey@uwyo.edu

Abstract. To support emerging database applications that deal with continuously changing (or moving) data objects (CCDOs), such as vehicles, RFIDs, and multi-stimuli sensors, one requires an efficient data management system that can store, update, and retrieve large sets of CCDOs. Although actual CCDOs can continuously change over time, computer systems cannot deal with continuously occurring infinitesimal changes. Thus, in the data management system, each object's spatiotemporal values are associated with a certain degree of uncertainty at virtually every point in time, and the queries are mostly processed over estimates characterizing the uncertainty. The smaller the uncertainty is, the better the query performance becomes. The paper proposes a sophisticated asymmetric uncertainty model, called the Tornado Model, which can effectively represent, process, and minimize the data uncertainty for a wide variety of CCDO database applications.

Keywords: spatiotemporal database, trajectory, uncertainty.

1 Introduction

An increasing number of emerging applications deal with a large number of *continuously changing* (or moving) *data objects* (CCDOs), such as vehicles, sensors, and mobile computers. For example, in earth science applications, temperature, wind speed and direction, radio or microwave image, and various other measures (e.g., CO₂) associated with a certain geographic region can change continuously. Accordingly, new services and applications dealing with large sets of CCDOs are appearing. In the future, more complex and larger applications that deal with higher dimensional CCDOs (e.g., a moving sensor platform capturing multiple stimuli) will become commonplace – increasingly complex sensor devices will continue to proliferate alongside their potential applications. Efficient support for these CCDO applications will offer significant benefit in many broader challenging areas including

mobile databases, satellite image analysis, sensor networks, homeland security, internet security, environmental control, and disease surveillance.

To support large-scale CCDO applications, one requires a data management system that can store, update, and retrieve CCDOs. Each CCDO has both multidimensional-temporal (i.e., 2 or 3D geographic space or other information dimensions that vary with time) properties representing its continuous trajectory in an information space-time continuum as well as non-temporal properties such as identification, associated phone number, and address. Importantly, although CCDOs can continuously move or change (thus drawing continuous trajectories in a space-time), computer systems cannot deal with continuously occurring infinitesimal changes – this would effectively require infinite computational speed and sensor resolution. Thus, each object's continuously changing attribute values (e.g., location, velocity, and acceleration) can only be discretely updated. Hence, they are always associated with a degree of uncertainty, especially when there is a considerable time gap between two updated points.

Over the past years, database research communities have mainly focused on representing the uncertainty of spatiotemporal CCDO locations (i.e., the position property of estimated CCDO states). In contrast, sensor technology has evolved to detect or approximate higher order derivatives (e.g., velocity and acceleration). For example, most GPS loggers in the market are now capable of recording highly accurate velocity (and even acceleration) values. Virtually any intelligent sensory device that can rapidly detect its stimuli can be used to produce the higher order derivatives at the sensor level. By properly utilizing these additional sensory inputs, we can possibly support CCDO queries referring to the higher order derivatives of the trajectories (e.g., report every CCDO that possibly had an acceleration within the given acceleration range at the given point in time). Importantly, these additional inputs can be utilized to minimize the uncertainty of a database trajectory.

The paper presents our research initiated on the following goals: first, provide a basis for developing an uncertainty model that can represent not only the position uncertainty but also the derivative uncertainties; second, investigate the problem of minimizing the uncertainty using the sensor-level derivatives.

The rest of this paper is organized as follows: Section 2 summarizes related techniques and models. Section 3 characterizes CCDOs. Sections 4 and 5 propose and verify our novel trajectory uncertainty model that fulfills the above goals. Section 6 concludes the paper and proposes some future work.

2 Related Work

Several application-specific models of uncertainty have been proposed. One popular uncertainty model is that, at any point in time, the location of an object is within a certain distance d , of its last reported location. If the object moves further than d , it reports its new location and possibly changes the distance threshold d for future updates [12]. Given a point in time, the uncertainty is a circle with a radius d , bounding all possible locations of the object.

Another model assumes that an object always moves along straight lines (linear routes). The location of the object at any point in time is within a certain interval, centered at its last reported location, along the line of movement [12]. Different CCDO trajectory models that have no uncertainty consideration are found in the literature [7]. These models make sure that the exact velocity is always known by

requiring reports whenever the object's speed or direction changes. Other models assume that an object travels with a known velocity along a straight line, but can deviate from this path by a certain distance [10, 11].

An important study on the issues of uncertainty in the recorded past trajectories (history) of CCDOs is found in [6]. Assuming that the maximum velocity of an object is known, they prove that all possible locations of the object during the time interval between two consecutive observations (reported states) lie on an error ellipse. A complete trajectory of any object is obtained by using linear interpolation between two adjacent states. That is, a trajectory is approximated by a sequence of connected straight lines, each of which connects two consecutively reported CCDO observations. By using the error ellipse, the authors demonstrate how to process uncertainty range queries for trajectories. The error ellipse defined and proved in [6] is the projection of a three-dimensional spatiotemporal uncertainty region onto the two-dimensional data space. Similarly, [4] represents the set of all possible locations based on the intersection of two half cones that constraint the maximum deviation from two known locations. It also introduces multiple granularities to provide multiple views of a moving object.

Our approach, a mechanism that explicitly leverages an understanding and characterization of uncertainty for a generalized case of the CCDO, offers an alternative construct that is suitable for higher dimensional data and that can produce minimally bounding spatiotemporal uncertainty regions for both past and future trajectories of CCDOs by taking into account the temporally-varying higher order derivatives, such as velocity and acceleration. We call this uncertainty model the *Tornado Uncertainty Model (TUM)* because, for each reported CCDO state, the model produces a tornado-shaped uncertainty region in the space-time.

3 Explication of CCDO

Before presenting our uncertainty model, we define “Continuously Changing Data Object” (CCDO) through a series of ontological abstractions. Table 1 represents our explication of CCDO.

Table 1. Multi-level abstraction of CCDO

Abstraction	Definition
<i>CCDO</i>	A CCDO is a data object consisting of one or more <i>trajectories</i> and zero, one, or more non-temporal properties.
<i>trajectory</i>	A trajectory consists of <i>dynamics</i> and <i>f:time</i> → <i>snapshot</i> , where <i>time</i> is a past, current, or future point in time.
<i>snapshot</i>	A snapshot is a probability distribution that represents the probability of every possible <i>state</i> at a specific point in time. Depending on the <i>dynamics</i> and update policies, the probability distribution may or may not be bounded.
<i>state</i>	A state is a point in a multidimensional information space-time of which time is one dimension. Each state associated with zero or more of the following optional properties: velocity (i.e., direction and speed of changes, the 1 st derivative), acceleration (the 2 nd derivative), and higher order derivatives.
<i>dynamics</i>	The dynamics of a state is a set of domains each of which represents all possible values corresponding to a certain property of the state.

Considering an observer who reports the state of a CCDO as often as possible, the trajectory drawn by the object is viewed as a sequence of connected segments in space-time, and each segment connects two consecutively reported states of the object. Examining of Table 1, one may observe the following: only a subset of *states* can be stored in the database, due to the fact that no database can be continuously updated. We call these stored states *reported states*. Each pair of consecutive reported states of a CCDO represent a single trajectory segment. The reported states are the factual known states of the CCDOs, and only these known states can be committed to the database. All possible in-between states and future states of the CCDOs are then interpolated and extrapolated on the fly when it is necessary (e.g., query processing, data visualization, index maintenance, and data management). Given the theoretical possibility of an infinite number of states between two factual states, a mathematical model and computational approach is required to efficiently manage the ‘in-between’ and ‘future’ states.

4 The Tornado Model

As discussed in Section 3, a CCDO consists of a set of conventional non-temporal attributes and one or more temporal attributes (i.e., a location anchor or boundary points) each of which can draw a trajectory over time in a multidimensional attribute space (e.g., geographic space and sensor stimuli space). Among the trajectory components defined in Table 1, only a subset of states, called reported states (*RS*), and a subset of dynamics, which we call known dynamics (*KD*) in this paper, can be stored in a database. Then the trajectory snapshots, which represent the uncertainty of the discretely recorded trajectory, are calculated when necessary. Therefore, in order to formally present our trajectory model, we first explicate a *database trajectory* (a discretely recorded CCDO trajectory stored in a database). As given in Definition 1, a database trajectory consists of a sequence of reported states and some known dynamics.

Definition 1. Database Trajectory

A database trajectory ***DBTRAJ*** in a $(d+1)$ -dimensional space-time with d data (or spatial) dimensions and one time dimension consists of a sorted set ***RS*** of n reported states and a sorted set ***KD*** of m known dynamics, where

- For all $i=0,..,n-1$, RS_i is a tuple $\langle P^{(0)}, P^{(1)}, \dots P^{(k-1)}, T, IME^{(0)}, IME^{(1)}, \dots IME^{(k-1)} \rangle$, where
 - For all $l=0,..,k-1$, $P^{(l)}$ is a d -dimensional vector representing the l^{th} derivative of the trajectory at T (e.g., $P^{(0)}$, $P^{(1)}$, and $P^{(2)}$ are, respectively, a d -dimensional location, velocity, and acceleration at T);
 - T is a specific time point at which the above state was observed (or sensed);
 - For all $l=0,..,k-1$, $IME^{(l)}$ is the domain of possible instrument-and-measurement errors (deviations from the real) associated with $P^{(l)}$.
- For all $j=0,..,m-1$, KD_j is a tuple $\langle D^{(0)}, D^{(1)}, \dots D^{(k)}, T \rangle$, where
 - For all $l=0,..,k$, $D^{(l)}$ is the domain of the l^{th} derivative of the trajectory at T ;
 - T is a specific time point at which the above domains are valid.
- For all $i=0,..,n-2$, $RS_i.T \leq RS_{i+1}.T$.
- For all $j=0,..,m-2$, $KD_j.T \leq KD_{j+1}.T$.

Considering the location $P^{(0)}$ to be the 0^{th} derivative, the value of k represents the number of derivatives reported to the database system. For example, for an

application wherein sensors can detect and report only the 3-dimensional geographic location of the object each time, d is set to 3 and k is set to 1. If the sensors can report not only locations but also velocities (i.e., $P^{(1)}$), k is set to 2.

Based on this database trajectory model (i.e., Definition 1), we define our uncertainty model (i.e., *snapshot* defined in Table 1) as given in Definition 2 to calculate the snapshots (uncertainty) of the trajectory given a time point t .

Definition 2. Snapshot (Uncertainty Region)

$\text{SNAPSHOT}^{(i)}(\text{DBTRAJ}, t)$ can be defined as follows:

$$\begin{cases} E^{(i)}(\text{RS}_{l-1}, t) \cap E^{(i)}(\text{RS}_l, t) & \text{if } \exists_l (\text{RS}_{l-1}.T \leq t \leq \text{RS}_l.T) \\ E^{(i)}(\text{RS}_{n-1}, t) & \text{if } \text{RS}_{n-1}.T < t \\ E^{(i)}(\text{RS}_0, t) & \text{if } \text{RS}_0.T > t \\ \emptyset & \text{otherwise (i.e., no reported state)} \end{cases}, \quad (1)$$

where RS_0 and RS_{n-1} are, respectively, the first and last reported states of trajectory DBTRAJ ; $E^{(i)}()$ is a function that takes a reported state rs and a time point t as input and produces a set of all possible i^{th} derivatives of the trajectory at t . The calculation of the snapshot falls in one of four cases: (1) t is between the times of two consecutive reported states (i.e., $\exists_l (\text{RS}_{l-1}.T \leq t \leq \text{RS}_l.T)$); (2) t is greater (later) than the last reported state (i.e., $\text{RS}_{n-1}.T < t$); (3) t is smaller (earlier) than the first reported state (i.e., $\text{RS}_0.T > t$); (4) DBTRAJ has no reported state.

As shown in Definition 2, one must define the estimation function series E in order to fully define this trajectory uncertainty (snapshots) model.

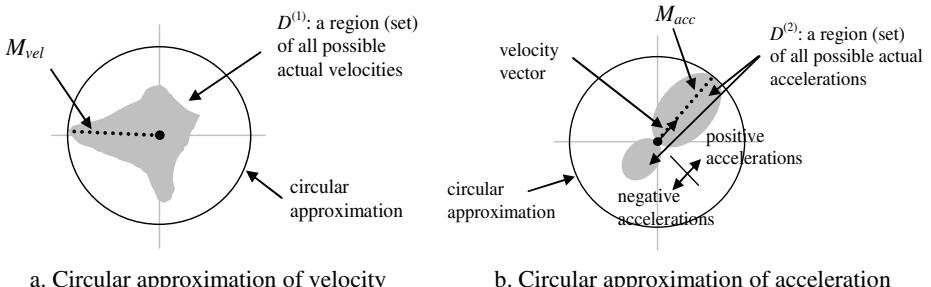


Fig. 1. Examples of the circular approximation

4.1 E of Degree 1: Revised Ellipse Model

Definitions 1 and 2 are generalized to accommodate any arbitrary set of instrument-and-measurement errors (IME) and dynamics (D) each of which can possibly be a highly complicated multidimensional shape. In addition, the model supports any level k of derivatives. In practical applications, the parameter k should be set to a specific value, and, for efficient calculation of trajectory snapshots, some form of approximation is necessary for both IME and D components. Let us first consider the following approximation: For every DBTRAJ ,

- (1) $k=1$;
- (2) $\forall i=0,\dots,n-1$, $RS_i.IME^{(0)}$ is a d -dimensional hyper-circle with a constant radius M_{err} ;
- (3) $\forall i=0,\dots,m-1$, $KD_i.D^{(0)}$ is a d -dimensional constant hyper-square $SPACE$;
- (4) $\forall i=0,\dots,m-1$, $KD_i.D^{(1)}$ is a d -dimensional hyper-circle with a constant radius M_{vel} .

Note that approximations (2) and (4) represent a circular approximation. Fig. 1a shows an example of the circular approximation of velocity. M_{vel} is the norm of the maximum possible actual velocity. Because of this approximation, possible velocities are independent of the location. Then, $E^{(0)}$ and $E^{(1)}$ can be defined as follows:

$$E^{(1)}(rs, t) = \{ p \mid \|p\| \leq M_{vel} \} \quad (2)$$

$$E^{(0)}(rs, t) = \{ p \mid p \in SPACE \wedge \exists_{iloc, ivel, t'} \left\{ \begin{array}{l} \|iloc - rs.P^{(0)}\| \leq M_{err} \wedge iloc \in SPACE \wedge \\ \|ivel\| \leq M_{vel} \wedge \\ 0 \leq t' \leq |t - rs.T| \wedge \\ p = iloc + ivel \cdot t' \end{array} \right\} \} \quad (3)$$

$E^{(1)}$ represents the hyper-circle of all possible velocities and $E^{(0)}$ represents all possible locations that the object, which starts with an initial location $iloc$ and any valid constant velocity $ivel$, can reach within the given time interval $|t - rs.T|$. As an example of this model, consider an object moving through one dimension of space over time. Fig. 2a shows an example of a trajectory segment connecting two reported states of the object. Let $RS_0 = \langle A, t_i, M_{err} \rangle$, $RS_1 = \langle B, t_j, M_{err} \rangle$ and let M_{vel} be the maximum change rate (i.e., the norm of the maximum possible velocity) of the CCDO. Then all possible states of the CCDO between t_i and t_j are bounded by the lines where $|\cot \theta| = M_{vel}$. The shaded region covers all possible locations of the object between t_i and t_j . The snapshot of the CCDO at any time point t that is between t_i and t_j is the cross section of this uncertainty region, produced by the cutting line at $time = t$. In this $d=1$ example, $SNAPSHOT^{(1)}$ is $[-M_{vel}, +M_{vel}]$ and $SNAPSHOT^{(0)}$ is

$$[A - M_{err} - M_{vel} \cdot |t - t_i|, A + M_{err} + M_{vel} \cdot |t - t_i|] \cap [B - M_{err} - M_{vel} \cdot |t - t_j|, B + M_{err} + M_{vel} \cdot |t - t_j|].$$

Similarly, when a CCDO continuously changes in a two-dimensional space, the snapshots between two consecutive reported states collectively represent the overlapping region of the two funnels (see Fig. 2b). It is important to note that, as shown in Fig. 2b, the projection of the snapshots onto the 2-dimensional data space is, in fact, the uncertainty ellipse that can be defined by the ellipse model [6] with a modification taking into account the instrument and measurement errors.

4.2 E of Degree 2: Tornado Uncertainty Model

As discussed in Section 3, in many applications, an accurate sensor-level approximation of trajectory derivatives is possible. For example, most GPS loggers

can record not only geographic positions but also corresponding velocity vectors. Hence, $k=2$ holds in the related CCDO applications. This section presents a specialization of the proposed uncertainty model with the following approximations in order to better support $k=2$ applications: For every *DBTRAJ*,

- (1) $k=2$;
- (2) $\forall_{i=0,\dots,n-1}$, $RS_i.IME^{(0)}$ is a d -dimensional hyper-circle with a constant radius $M_{err}^{(0)}$;
- (3) $\forall_{i=0,\dots,n-1}$, $RS_i.IME^{(1)}$ is a d -dimensional hyper-circle with a constant radius $M_{err}^{(1)}$;
- (4) $\forall_{i=0,\dots,m-1}$, $KD_i.D^{(0)}$ is a d -dimensional constant hyper-square *SPACE*;
- (5) $\forall_{i=0,\dots,m-1}$, $KD_i.D^{(1)}$ is a d -dimensional hyper-circle with a constant radius M_{vel} ;
- (6) $\forall_{i=0,\dots,m-1}$, $KD_i.D^{(2)}$ is a d -dimensional hyper-circle with a constant radius M_{acc} .

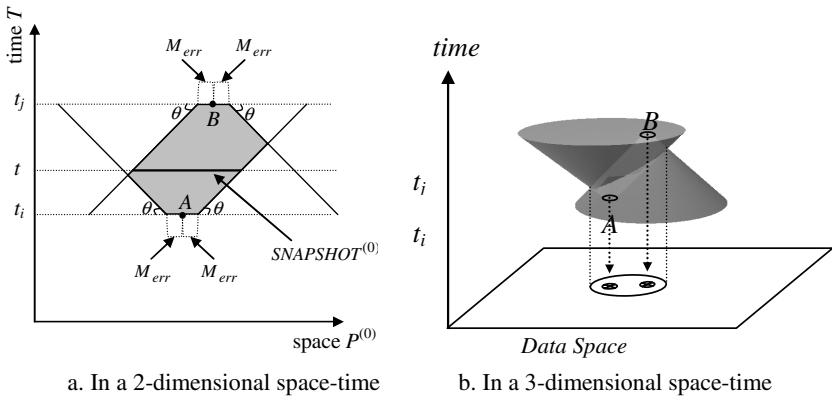


Fig. 2. Spatiotemporal uncertainty representing a trajectory segment

Note that this approximation includes a circular approximation of accelerations as shown in Fig. 1b. M_{acc} represents the norm of the maximum possible actual acceleration. Because we use this circular approximation, possible accelerations are independent of the corresponding velocity vector. That is, $E^{(2)}$ is defined as Eq. 4. Then, the estimation functions $E^{(1)}$ and $E^{(0)}$ can be defined as Eqs. 5 and 6:

$E^{(2)}$ represents the constant hyper-circle of all possible accelerations and $E^{(1)}$ represents all possible velocities that the object, which starts with the initial velocity *ivel* and a valid acceleration *iacc*, can reach within the given time interval $|t-rs.T|$. Then, $E^{(0)}$ defines all possible locations the object can reach.

In the last condition of Eq. 6, *iloc*, *ivel*, and *iacc* represent a possible initial location, velocity, and acceleration at *rs.T*. The integral term represents a cumulative location displacement while the object is constantly accelerating at the rate of *iacc*, and the last term represents the location displacement that can be produced by a fixed velocity *ivel+iacc*t'* during $|t-rs.T| - t'$. Note that, unlike the symmetric estimation $E^{(1)}$, $E^{(0)}$ needs to compare *t* and *rs.T*. When $E^{(0)}$ estimates a future snapshot (i.e., $t \geq rs.T$), possible displacements are added to the initial location *iloc* (i.e., adding future possible displacements); for past estimations, possible displacements are subtracted from *iloc* (i.e., canceling out past possible displacements). Note that Eqs. 3 and 5 do not need this differentiation because, given a single reported state *rs*, their snapshots

are symmetric to the d -dimensional hyper-plane that is perpendicular to the time axis at $rs.T$ (i.e., the future snapshots are the past snapshots). The maximum directional displacement in location (i.e., the maximum distance the object can travel in a certain direction during $|t-rs.T|$) is obtained when $iacc$ is a boundary point of the acceleration circle $E^{(2)}(rs,t)$ and $ivel+iacc \cdot t'$ is a boundary of the velocity circle $E^{(1)}(rs,t')$.

Because the symmetric circular approximation of acceleration (resp., velocity) fully encloses all possible actual accelerations (resp., velocities), no real object can go beyond the boundary of $E^{(2)}$, $E^{(1)}$, or $E^{(0)}$.

$$E^{(2)}(rs,t) = \{ p \mid \|p\| \leq M_{acc} \} \quad (4)$$

$$E^{(1)}(rs,t) = \{ p \mid \|p\| \leq M_{vel} \wedge \exists_{ivel,iacc,t'} \left. \begin{array}{l} \|ivel - rs.P^{(1)}\| \leq M_{err}^{(1)} \wedge \|ivel\| \leq M_{vel} \\ \|iacc\| \leq M_{acc} \wedge \\ 0 \leq t' \leq |t - rs.T| \wedge \\ p = ivel + iacc \cdot t' \end{array} \right\} \quad (5)$$

$$E^{(0)}(rs,t) =$$

$$\left. \begin{array}{l} \|iloc - rs.P^{(0)}\| \leq M_{err}^{(0)} \wedge iloc \in SPACE \wedge \\ \|ivel - rs.P^{(1)}\| \leq M_{err}^{(1)} \wedge \|ivel\| \leq M_{vel} \wedge \\ \|iacc\| \leq M_{acc} \wedge \\ 0 \leq t' \leq |t - rs.T| \wedge \\ \{ p \mid p \in SPACE \wedge \exists_{ivel,iacc,t'} ivel + iacc \cdot t' \in E^{(1)}(rs,t') \wedge \\ p = \begin{cases} iloc + \int_0^{t'} (ivel + iacc \cdot T) dT + (ivel + iacc \cdot t') \cdot (|t - rs.T| - t') & \text{if } t \geq rs.T \\ iloc - \int_0^{t'} (ivel + iacc \cdot T) dT - (ivel + iacc \cdot t') \cdot (|t - rs.T| - t') & \text{otherwise} \end{cases} \end{array} \right\} \quad (6)$$

For an example, let us assume that a CCDO (a car) moves in two dimensional space from RS_0 (located at $x=0.0$ and $y=0.0$ at time 0) to RS_1 (located at $x=-1.5$ and $y=655.80$ at time 20) with an initial velocity $ivel$ (0.083 meters per second along x axis and 32.34 m/s along y axis). The maximum velocity and acceleration of the car are M_{vel} (50 meters per second) and M_{acc} (2.78 m/s per second), respectively.

Step 1: From RS_0 , calculate the maximum possible displacements $E^{(0)}(RS_0, t)$ of the CCDO in all directions over time $0 \leq t \leq 20$. First, we calculate all possible accelerations $E^{(2)}$ using the circular approximation. To discretely represent the boundary of the hyper-circle of $E^{(2)}$, one can choose a certain number of discrete points¹ along the boundary of the hyper-circle with a fixed interval. Then, these points represent the set of all possible maximum accelerations. Then $E^{(1)}$ and $E^{(2)}$ can be calculated by Eq. 5 and Eq. 6, respectively. Fig. 3 example shows the calculated results for $E^{(0)}(RS_0, 6)$. The same process is applied to calculate $E^{(0)}(RS_1, t)$ from RS_1 .

¹ The more points we use to create the polygons the more accurate the estimation of the uncertainty region is. However, it is not practical to use too many points at each time step since it takes a lot of computing time. For our experiments in Sec. 5, we used 100 points.

Step 2: Two polygons can be created by connecting adjacent points in $E^{(0)}(RS_0, t)$ and $E^{(0)}(RS_1, t)$, respectively. We use the Graham's algorithm [5], which finds the convex hull of a given set of points. The two polygons in Fig. 3b represent the maximum displacements (boundaries) from the two locations, one from RS_0 and the other from RS_1 at any time t between 0 and 20.

Step 3: Quantify the overlapping area of the two intersecting polygons at time t . First, we used the *ray drawing and crossing number algorithm* [5] for each boundary point of one polygon against the other polygon to see if the point is common. Second, the set of points that are common (i.e., overlapping points) were used to create another convex polygon using the *Graham's algorithm*, which represents the overlapping area of $E^{(0)}(RS_0, t)$ and $E^{(0)}(RS_1, t)$ (shaded area in Fig. 3b). Finally, we use the following formula to calculate the area of the uncertainty region:

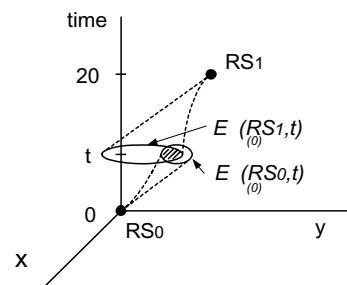
$$A(\text{pol}) = \sum_{i=0}^{n-1} (x_i y_{i+1} - y_i x_{i+1}) / 2, \text{ where } A(\text{pol}) \text{ is the area of the polygon } \text{pol}, \text{ and } x_i, y_i \text{ are the coordinates of a point in } \text{pol}.$$

Step 4: We repeated Steps 1-3 as a function of time, for example every second, to quantify the overlapping area of the two intersecting polygons at a certain time. The summation of all the areas over the whole interval, from time 0 to 20, is the uncertainty volume.

Similarly, the uncertainty volume of the revised ellipse model can be quantified using Eqs. 2 and 3.

$E^{(2)}$		$E^{(1)}$		$E^{(0)}$	
x	y	x	y	x	y
1.8	1.3	12.7	41.5	32.0	217.3
0.7	2.1	4.90	44.4	12.5	232.4
-0.7	2.1	-4.7	44.4	-11.5	232.4
-1.8	1.3	-12.5	41.5	-31.0	217.3
-2.2	-0.0	-15.5	32.3	-38.5	194.4
-1.8	-1.3	-12.5	23.2	-31.0	171.5
-0.7	-2.1	-4.7	17.6	-11.5	157.3
0.7	-2.1	4.9	17.6	12.5	157.3
1.8	-1.3	12.7	23.2	32.0	171.5
2.2	0.0	15.6	32.3	39.5	194.4

a. Calculation of $E^{(i)}(RS_0, t)$



b. Illustration

Fig. 3. Overlapping region at time t

5 Experiments

Using a portable GPS device (Trimble Navigation's ProXRS Receiver with GPS logger), which can record a pair <location-time, velocity> every second, we collected real GPS data. Every report was 3-dimensional (i.e., longitude, latitude, and time). We placed the GPS device in a car and drove from a location in the north of Denver, Colorado, to Loveland, Colorado along the interstate highway 25. Every second, we logged spatiotemporal data from the GPS device. Our collected data include both relatively straight movement on a highway and some winding movement in a city

area, which is useful for a better comparison. For the comparison between the two models, we created trajectories based on the logged records. A time interval T_{int} defines the elapsed real time between two selected adjacent records. First we selected the logged records with a fixed 20-second time interval (i.e., $T_{int}=20$) and we also randomly selected the records with a sampling ratio of about 5%.

In all experiments, the maximum velocity, M_{vel} , was set to 180km/hour (50meters/sec). The maximum acceleration, M_{acc} , was set to 10km/hour per second (2.78 meters/sec per second). The maximum report (instrument and measurement) errors $M_{err}^{(1)}$ and $M_{err}^{(0)}$ were set to 0 and 2 meters, respectively. For the circular approximation discussed in Sec. 4.2, we selected 100 points along the boundary of the acceleration hyper-circle with a fixed interval.

First, using the real GPS data selected with a fixed time interval ($T_{int} = 20$ seconds), we constructed 33 reported states (RS) and quantified the uncertainty region volume between each two adjacent states following the steps in Sec. 4.2. Fig. 4a shows the quantified uncertainty volumes of the two models. Fig. 4b shows the percentage reduction in uncertainty volume between the tornado uncertainty model (TUM) and the revised ellipse model (REM). On the average, TUM produced 94% of reduction compared to REM. The first 20 reported states were collected while driving on a

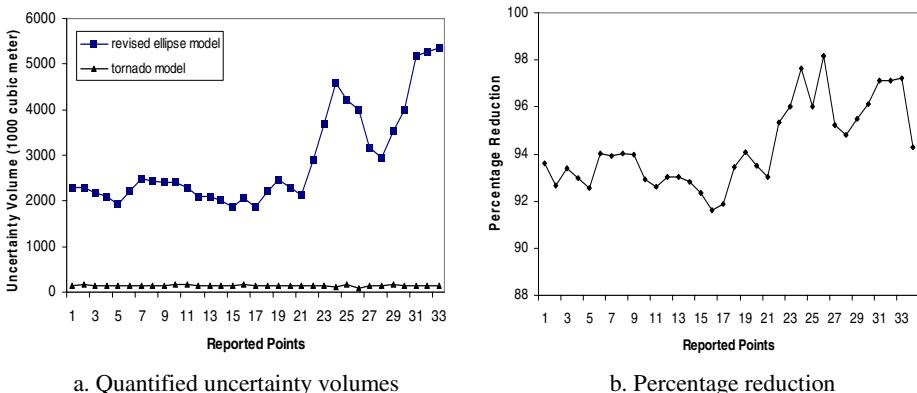


Fig. 4. Comparison of two models, with 20 sec fixed interval

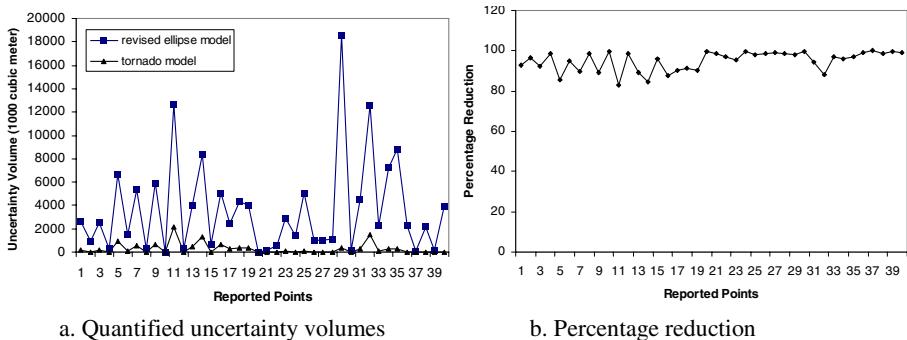


Fig. 5. Comparison of two models, with random interval

straight highway with a high velocity and the last 13 reported states were collected while driving in a city area with a low velocity. The results show that the volume difference between TUM and REM becomes greater when the object moves slowly.

Next, we repeated the same experiment with randomly selected records by generating arbitrary intervals between 5 and 35 seconds ($5 \leq T_{int} \leq 35$). Figs. 5a and 5b show the quantified uncertainty volumes of TUM and REM and the percentage difference in uncertainty volume between the two models, respectively. TUM produced a 95% reduction compared to REM on the average.

All results show that the uncertainty volumes produced by TUM are significantly smaller than their counterparts produced by REM. To investigate how efficient TUM is over REM, we performed the following experiments by varying a couple of factors. First, assuming identical initial velocity $ivel$, maximum velocity M_{vel} , and maximum acceleration M_{acc} , we quantified the uncertainty volumes with varying T_{int} between reported states. Table 2a shows that TUM becomes more efficient compared to REM as T_{int} gets smaller. Second, assuming identical $ivel$, M_{vel} , and T_{int} , we quantified the uncertainty volumes with varying M_{acc} . Table 2b shows that TUM becomes more efficient compared to REM as M_{acc} gets smaller.

As shown in Eqs. 3 and 6, $E^{(0)}$ is a function of $ivel$, M_{vel} , M_{acc} , and the elapsed time t from a reported state. $E^{(0)}$ gets more dispersed as the velocity of the object approaches M_{vel} . Eq. 3 (i.e., REM) produces more dispersion of $E^{(0)}$ than Eq. 6 (i.e., TUM) because REM assumes that M_{vel} is possible during the whole interval $|t-rs,T|$ (i.e., $t' = |t-rs,T|$). However, by considering possible accelerations and the reported velocity, TUM gradually increases the velocity from $ivel$ to M_{vel} over time (i.e., the integral term of Eq. 6), which also gradually increases the dispersion of $E^{(0)}$. Thus, the slower the velocity reaches the maximum (i.e., as the maximum possible t' increases), the smaller the dispersion of $E^{(0)}$ becomes. The difference of uncertainty volumes between REM and TUM becomes accordingly wider. This is the reason why TUM becomes more efficient as M_{acc} and/or $ivel$ decreases. Similarly, TUM becomes more efficient when the time interval is shorter. The velocity may not even reach to the maximum when the time interval is short.

Table 2. The average percentage reduction of uncertainty volume

T_{int} Range in Seconds	Average % Reduction
5-10	99.25
11-15	98.41
16-20	96.40
21-25	93.94
26-30	88.22
30-35	87.71

a. Varying time interval ($M_{acc}=10$)

Max. Acceleration (M_{acc})	Average % Reduction
10	94.30
20	81.79
30	69.26
40	58.90
50	51.66
60	46.30

b. Varying M_{acc} ($T_{int}=20$ seconds)

6 Conclusion

In this paper, we proposed a novel and practical framework for managing multidimensional CCDOs (i.e., spatiotemporal trajectory representation and processing). The *Tornado* model can more efficiently support both conventional

CCDOs that move in a 2- or 3-dimensional geographic space and emerging high-dimensional CCDOs, such as combined sensor streams and satellite data. Based on the framework, one might be able to devise an uncertainty model that employs a more precise approximation of the actual derivatives.

Processing a query with uncertainty means that each result data item is associated with the probability (or likelihood) that the item really satisfies the query predicates [3]. To support probabilistic query processing, one needs to calculate the probability density of each snapshot: An appropriate application-specific distribution (e.g., skewed-normal random distribution) can be used to estimate the probability density [1,2,9]. [13] provides some non-linear methods that can significantly reduce the errors associated with the peak point of the probability density. If the snapshots can be further minimized, the spatiotemporal regions requiring indexing can also be commensurately limited and the query results will be associated with more probable likelihoods. By taking into account how the environment may be variably constraining movement and thus variably affecting the set of possible states of the CCDO, one can further reduce the snapshots through a separate processing steps of contextualizing (modifying) the probability distributions [8].

In the two-phase (filtering-refinement) query processing, the smaller the uncertainty regions are, the lower the rate of false-drops (i.e., the objects that are selected in the filtering-phase but discarded in the refinement-phase) becomes. On the other hand, the computation cost of the uncertainty model affects the cost of the refinement step of the query processing. These two phases are not independent, since the false-drop rate of the first phase determines the number of objects to be tested in the refinement step. We have also considered the average cost of testing a candidate in the refinement step as follows: In our discrete implementation of the uncertainty models and experiments using a Linux machine equipped with an Intel Pentium III 800MHz and 256MB main memory space, the degree-1 Tornado model (REM) took about 0.7-0.8 microseconds of CPU time to interpolate each boundary point of the uncertainty region, and the degree-2 Tornado model (TUM) required about 5 microseconds. On the other hand, compared to REM, TUM reduced the uncertainty volumes by more than an order of magnitude on average. Understanding the implication of these in actual query performance requires the hardware platform, adopted cache-buffering method, the trajectory data, and the access method.

References

1. Azzalini A., Capitanio, A. Statistical applications of the multivariate skew-normal distribution. *Journal of the Royal Statistical Society, Series B*(61), 1999, 579-602.
2. Azzalini A., Valle, A. D. The multivariate skew-normal distribution. *Biometrika*, 83, 1996, 715-726.
3. Cheng, R., Kalashnikov, D., Prabhakar, S. Evaluating Probabilistic Queries over Imprecise Data. *IEEE Transactions on Knowledge and Data Engineering*, 16(9), 2004, 1112-1127.
4. Hornsby, K., Egenhofer, M. J. Modeling Moving Objects over Multiple Granularities, *Annals of Mathematics and Artificial Intelligence*, 36(1-2), 2002, 177-194.
5. O'Rourke, J. Computational Geometry in C, 2nd ed., Cambridge University Press, 1998.
6. Pfoser, D., Jensen, C. S. Capturing the Uncertainty of Moving-Objects Representations. *In Proc Int. Conf. on Scientific and Statistical Database Management*, 1999, 123-132.

7. Pfoser, D., Jensen, C. S. Querying the Trajectories of On-Line Mobile Objects. In *Proc. ACM MobiDE International Workshop on Data Engineering for Wireless and Mobile Access*, 2001, 66-73.
8. Prager, S. D.: Environmental Contextualization of Uncertainty for Moving Objects. In: *Proc. GeoComputation*. Ann Arbor, Michigan, 2005.
9. R Development Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2004.
10. Sistla, P. A., Wolfson, O., Chamberlain, S., Dao, S. Querying the Uncertain Position of Moving Objects. *Temporal Databases: Research and Practice*, 1997, 310-337.
11. Trajcevski, G., Wolfson, O., Zhang, F., Chamberlain, S. The Geometry of Uncertainty in Moving Object Databases. In *Proc. Int'l Conf. on Extending Database Technology*, 2002, 233-250.
12. Wolfson, O., Sistla, P. A., Chamberlain, S., Yesha, Y. Updating and Querying Databases that Track Mobile Units. *Distributed and Parallel Databases*, 7(3), 1999, 257-387.
13. Yu, B., Kim, S. H., Bailey, T., Gamboa, R. Curve-Based Representation of Moving Object Trajectories. *IEEE International Database Engineering and Applications Symposium*, 2004, 419-425.

ClusterShedy: Load Shedding Using Moving Clusters over Spatio-temporal Data Streams

Rimma V. Nehme¹ and Elke A. Rundensteiner²

¹ Purdue University, West Lafayette, IN 47907 USA

² Worcester Polytechnic Institute, Worcester, MA 01608 USA

rnehme@cs.purdue.edu, rundenst@cs.wpi.edu

Abstract. Moving object environments are characterized by large numbers of objects continuously sending location updates. At times, data arrival rates may spike up, causing the load on the system to exceed its capacity. This may result in increased output latencies, potentially leading to invalid or obsolete answers. Dropping data randomly, the most frequently used approach in the literature for load shedding, may adversely affect the accuracy of the results. We thus propose a load shedding technique customized for spatio-temporal stream data. In our model, spatio-temporal properties, such as location, time, direction and speed over time, serve as critical factors in the load shedding decision. The main idea is to abstract similarly moving objects into *moving clusters* which serve as summaries of their members' movement. Based on resource restrictions, members within clusters may be selectively discarded, while their locations are being approximated by their respective moving clusters. Our experimental study illustrates the performance gains achieved by our load-shedding framework and the tradeoff between the amount of data shed and the result accuracy.

1 Introduction

Applications dealing with extremely large numbers of moving objects are becoming increasingly common. These include fleet monitoring [31], location-based services [18] and scientific applications [25]. In such applications, queries are typically continuously evaluated over data streams composed of location updates. At times such data streams may become bursty and thus exceed system capacity.

However, existing load smoothing techniques [16][24][30] that store the tuples that cannot be processed into archives (spill them to disk) are not viable options for streaming spatio-temporal data. This is because spatio-temporal applications typically have real-time response requirements [18][21]. Any delay in the answer to a query would give obsolete results, and with objects continuously changing their locations, make them invalid or useless.

In order to deal with resource limitations in a graceful way, returning approximate query answers instead of exact answers has emerged as a promising approach [3][8][9]. Load shedding is a popular method to approximate query answers for stream processing while reducing the consumption of resources. The

goal is to minimize inaccuracy in query answers while keeping up with the incoming data load. The current state-of-the-art in load shedding [12, 9, 22, 26, 28, 29] can be categorized into two main approaches. The first relies on syntactic (random) load shedding, where tuples are discarded randomly based on expected system performance metrics such as output rate [9, 27]. The second approach, also known as semantic load shedding, assigns priorities to tuples based on their utility (semantics) to the application and then sheds those with low priority first [6, 27]. However, both of these approaches may suffer from high inaccuracy if applied to spatio-temporal data streams. The reason is two-fold: (1) they do not consider the spatio-temporal properties of the moving objects when deciding which data to load shed, and (2) they do not consider that both queries as well as objects have the ability to change their locations. Hence the results to queries depend on both the *positions of the moving objects* and of *the moving queries* at the time of querying.

To motivate the solution presented in this paper, we consider a scenario from the supply-chain management application – fleet monitoring. We assume that vehicles are equipped with positioning devices (e.g., GPS) and are travelling in convoys, i.e., in close proximity from each other. Using random load shedding, all vehicles' location updates are treated equally. Thus any tuple is equally likely to be discarded and the whereabouts of the vehicle may be unknown for some duration of time. Using semantic load shedding, a user may specify vehicles with the most valuable (e.g., expensive or perishable) cargo having the highest utility. The locations of the vehicles with lower utility may be discarded first, and thus temporarily loosing the location information of those moving objects.

The scenario above illustrates that using current load shedding techniques, the spatio-temporal properties of the moving objects are not taken into account when deciding which data to discard. However, if the workload must be reduced by dropping some data, taking into account such spatio-temporal properties as location, speed, direction, much higher accuracy can be achieved. Another point the scenario above illustrates is the spatio-temporal relationship of several different objects relative to each other, more specifically the *similarity of movement*. Thus intuitively if we can approximate similarly moving objects into clusters, and keep track of spatio-temporal properties of the cluster as a whole, then we could load shed the objects close to the center of the cluster without losing much in the results accuracy. So the decision to discard certain data is not related to only one object, but rather to dynamically formed sets of objects. To the best of our knowledge, no prior work has addressed this thus far.

1.1 Spatio-temporal Similarity

We observe that large numbers of moving objects often share some spatio-temporal properties, in fact, they often naturally move in clusters for some periods of time [7, 14]. For example, migrating animals, city pedestrians, or a

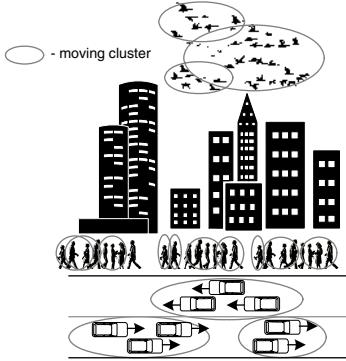


Fig. 1. Conceptual View of Moving Clusters

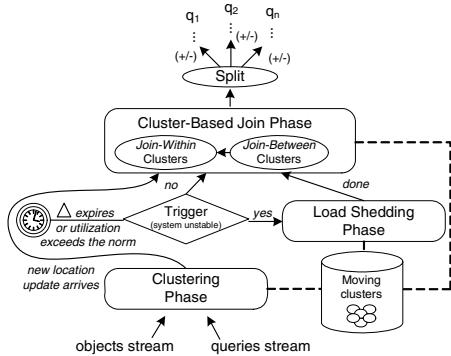


Fig. 2. ClusterSheddy Architecture

convoy of cars that follow the same route in a city naturally form moving clusters (Fig. 1). Such moving clusters do not always retain the same set of objects for their lifetime, rather some objects may enter or leave over time. For example, new animals may enter the migrating group, and others may leave the group (e.g., animals attacked by predators). While belonging to a particular cluster, the object shares *similar properties* with the other objects that belong to the same cluster. In this case, the spatio-temporal properties of the cluster such as speed, direction¹, and relative proximity of other moving objects summarize *how* these objects are moving and *where* they currently are.

A cluster can in some sense serve as a “common-feature-abstraction” of a group of moving objects sharing spatio-temporal properties. We now postulate that this abstraction can be exploited for efficient load shedding.

1.2 Our Contributions: ClusterSheddy Framework

In this paper, we present the ClusterSheddy framework for processing spatio-temporal queries on moving objects. ClusterSheddy is equipped with novel methods for spatio-temporal load shedding based on motion semantics. Moving clusters, abstracting similarly moving objects and queries, serve as summaries of their members and preserve their location, even if approximate, when their individual positions are load shed. The novelty of our method is that it uses dynamic clusters, together with the knowledge of the current system resources to determine *when*, *how much* and *which* data to load shed. Inside each moving cluster, a nested data structure, termed *nucleus*, abstracts the positions of the cluster members whose positions are load shed. In other words, the load shedding in ClusterSheddy takes a “from-inside-out” approach, where objects/queries closest to the center of the cluster are load shed first. The motivation is that

¹ We measure direction using a counterclockwise angle of rotation with due East.

Using this convention, a vector with a direction of 30 degrees is a vector which has been rotated 30 degrees in a counterclockwise direction relative to due east.

the closer cluster members are to the centroid, the more accurately the cluster approximates their individual locations. The sizes of the moving clusters' nuclei are resource-sensitive, meaning that the nucleus size of a moving cluster changes depending on the current resource availability. We measure the quality of a load shedding policy in terms of the deviation of the estimated answers produced by the system from the actual answers. Experimental results illustrate that ClusterSheddy is very effective in quickly reducing the load while maintaining good accuracy of the results.

Roadmap: Section 2 provides an overview of the ClusterSheddy framework and the moving cluster abstractions over spatio-temporal streams. Section 3 describes the load shedding technique based on moving clusters and the different policies for shedding clusters. Section 4 presents our experimental results. Section 5 discusses related work, and Section 6 concludes the paper.

2 ClusterSheddy Framework

2.1 Query Evaluation in ClusterSheddy

ClusterSheddy is encapsulated into a physical non-blocking pipelined query operator that can be combined with traditional operators in a query network plan. The input to ClusterSheddy consists of moving objects and spatio-temporal query streams. Moving objects' location updates arrive in the form (oid, Loc, t) , where oid is an object id, and Loc is its location at time t . Continuous queries arrive in the form $(qid, Loc, t, qType, qAttrs)$, where qid is a query id, and Loc is its location at time t , $qType$ is a query type (e.g., knn, range), and $qAttrs$ represents query attributes (e.g., a value of k for a knn query).

The ClusterSheddy execution process consists of three phases: (1) clustering, (2) cluster-based join, and (3) load shedding (Fig. 2). When new location data for an object/query arrives, the object/query joins either an existing cluster or forms its own cluster (clustering phase). Similar to our prior work, SCUBA [20], spatio-temporal queries on moving objects are evaluated by first performing a spatial *join between* moving clusters pruning true negatives. If two clusters do not intersect with one other, the objects and queries belonging to these clusters are guaranteed to not join either. Thereafter, in the *join-within* step, individual objects and queries inside the clusters are joined with each other. This two-step filter-and-join process helps reduce the number of unnecessary spatial joins.

Unlike SCUBA, ClusterSheddy implements incremental query evaluation and unlike SINA [19] and SEA-CNN [32], it is done at the coarser level of moving clusters rather than of individual objects and queries. Such incremental approach helps to avoid continuous re-evaluation of spatio-temporal queries. Moreover, ClusterSheddy effectively employs load shedding based on moving clusters, a task not addressed in these prior works.

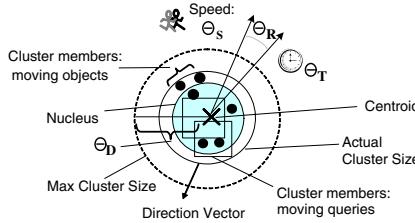


Fig. 3. A Moving Cluster Representation

Notation	Property	Description
$m.Cid$	Cluster ID	Numeric cluster identifier
$m.oCount$	Object Members	Number of moving objects
$m.qCount$	Query Members	Number of moving queries
$m.Loc$	Centroid Location	Location of the centroid of the cluster
$m.R$	Radius	Radius of the cluster
$m.AveSpeed$	Average Speed	Average of the speeds of all cluster members
$m.AveDir$	Direction	Movement Direction
$m.UpdTime$	Update Time	Last update/check time
$m.MaxValue$	Max Size	Maximum area that the cluster can increase up to
$m.Size$	Actual Size	Current area of the cluster, $m.Size = \pi m.R^2$
$m.Answers$	Results	Query results associated with the queries in the cluster

Fig. 4. Moving Cluster Properties

2.2 Moving Cluster Abstraction

Given the intuition highlighted in Section 1.1 that the moving objects often travel closely together in space for some period of time, we group moving entities² into *moving clusters* based on their shared spatio-temporal properties (Fig. 3). Moving entities that don't satisfy conditions of any existing clusters form their own clusters. When moving entities change their spatio-temporal properties, they may enter or leave a moving cluster, and the properties of that cluster (depicted in Fig. 4) are then adjusted accordingly.

Four similarity thresholds play a key role in determining the moving clusters³. Using these thresholds, we define the *similarity* among moving entities.

Definition 1. (*Similarity Condition*) Let Θ_S be the maximum speed difference threshold, Θ_D the maximum spatial distance threshold, Θ_R the maximum direction difference threshold, and Θ_T the maximum time difference threshold. Let t_k and t_l be the times when moving entities e_i and e_j ($i \neq j$) last updated their spatio-temporal properties. Then if $|e_i.Speed - e_j.Speed| \leq \Theta_S$,

² By moving entities we mean both moving objects and spatio-temporal queries.

³ Deriving threshold values that give you near-optimal clustering is a research area of its own. In our work, we approximated the threshold values that would cluster on average a certain number of objects per cluster based on the properties of the data [12].

```

PROCEDURE ClusterMovingObject (MovingCluster M)
1. new object o arrives
2. if distance between o and M.Loc < distance threshold
3.   and speed difference between o and M < speed threshold
4.   and direction difference between o and M < direction threshold
5.   and time difference between o location update and M update < time threshold
6.     add object o to the moving cluster M, updating cluster properties:
        a) member object count (M.Count) and total member count (M.Count)
        b) average speed (M.AveSpeed)
        c) average direction (M.Dir)
        d) location of the cluster centroid (M.Loc)
        e) cluster radius (M.R)
        f) record the time of the cluster update (M.UpdTime)
7.   else create new cluster based on the properties of object o

```

Fig. 5. Pseudo Code for Clustering Moving Objects

and $|e_i.Loc - e_j.Loc|^4 \leq \Theta_D$, and $|e_i.Dir - e_j.Dir| \leq \Theta_R$, and $|t_k - t_l| \leq \Theta_T$, the entities are said to be similar, $e_i \stackrel{s}{=} e_j$.

Definition 2. (Moving Cluster) Let $E = \{e_1, e_2, \dots, e_i\}$ be a set of moving entities. A moving cluster m is a non-empty subset of E ($m \subseteq E$), with spatio-temporal properties $m_{spt} = (\text{AveSpeed}, \text{AveDir}, \text{Loc}, R, t, \dots)$ which represents the average of the spatio-temporal properties of all entities $e_i \in m$, and where each e_i satisfies the similarity condition with respect to m_{spt} .

Using Definition 1 above, a moving entity e_i that is found to be in close proximity of a cluster centroid ($m.Loc$) and has similar properties with the cluster, is added to that respected moving cluster. Our clustering method is based on the classic *leader-follower* (*LF*) algorithm⁵ [10][11]. The *LF* algorithm can handle incrementally streaming data, producing adequate quality moving clusters in linear time. Fig. 5 gives the pseudo-code for clustering moving objects. Similar processing is done for clustering moving queries. Due to space constraints, we omit the description of the clustering procedure. For more details, we refer the reader to [20].

2.3 Incremental Query Evaluation Using Moving Clusters

ClusterSheddy uses an incremental strategy in evaluating joins *between* moving clusters, and then *within* the overlapping clusters that we describe next.

Incremental Join-Between: Consider two moving clusters m_1 and m_2 (Fig. 6). When performing a join *between* moving clusters, ClusterSheddy distinguishes between four cases as illustrated in Table 1⁶. Column “Clusters at time t_0 ” indicates if m_1 and m_2 intersected at an old time t_0 , column “Clusters at time t_1 ”

⁴ The difference is measured by a distance function (e.g., euclidean distance).

⁵ However, any alternative clustering algorithm can also be plugged-in, as long as it is efficient.

⁶ By \cap , we denote intersection.

Table 1. Cases for Incremental Evaluation: *Join-Between* Moving Clusters

Case	Clusters at time t_0	Clusters at time t_1	Join-Within Needed	Result Updates	Illustration
1.	$m_1 \cap m_2 = \emptyset$	$m_1 \cap m_2 = \emptyset$	-	-	Fig. 6a
2.	$m_1 \cap m_2 = \emptyset$	$m_1 \cap m_2 \neq \emptyset$	✓	positive	Fig. 6b
3.	$m_1 \cap m_2 \neq \emptyset$	$m_1 \cap m_2 = \emptyset$	-	negative	Fig. 6c
4.	$m_1 \cap m_2 \neq \emptyset$	$m_1 \cap m_2 \neq \emptyset$	✓	positive/negative	Fig. 6d

Table 2. Cases for Incremental Evaluation: *Join-Within* Moving Clusters

Case	Clustering at time t_1	$m.Ans$ at time t_0	o, q at t_1	Update Answer Set	Result Updates	Illustration
1.	$o \in m$ and $q \in m$	$(q, o, m.Cid) \in m.Ans$	$o \in q$	-	-	Fig. 7a
2.	$o \in m$ and $q \in m$	$(q, o, m.Cid) \in m.Ans$	$o \notin q$	$(q, o, m.Cid) \notin m.Ans$	$(q, -o)$	Fig. 7b
3.	$o \in m'$ and $q \in m$	$(q, o, m.Cid) \in m.Ans$	$o \in q$	$(q, o, m'.Cid) \in m.Ans$	-	Fig. 7c
4.	$o \in m'$ and $q \in m$	$(q, o, m.Cid) \in m.Ans$	$o \notin q$	$(q, o, m.Cid) \notin m.Ans$	$(q, -o)$	Fig. 7d
5.	$o \in m'$ and $q \in m$	$(q, o, m.Cid) \notin m.Ans$	$o \in q$	$(q, o, m'.Cid) \in m.Ans$	$(q, +o)$	Fig. 7e
6.	$o \in m'$ and $q \in m$	$(q, o, m.Cid) \notin m.Ans$	$o \notin q$	-	-	-

describes if they are currently intersecting (at time t_1). “*Join-Within Needed*” column specifies if *join-within* needs to be performed after this current *join-between*, while the “*Results Updates*” column describes the types of result updates that would be sent to the output stream. Column “*Illustration*” names the figure that graphically illustrates this case. For example, consider Case 1. If at time t_0 and at time t_1 two clusters m_1 and m_2 are not overlapping, no further processing is needed and no result updates are sent.⁷

Incremental Join-Within: To illustrate incremental *join-within* moving clusters, consider a query q which belongs to a cluster m and a moving object o which at time t_0 was in the same cluster m as q , but at time t_1 may belong to either the same cluster (m) or to any moving cluster currently intersecting with m (e.g., m') (Fig. 7). ClusterShddy, at time t_1 , distinguishes among six cases (Table 2)⁸. Column “*Clustering at time t_1* ” describes which moving clusters the object o and query q belong to at time t_1 . “ *$m.Ans$ at time t_0* ” column describes the result set associated with cluster m for query q . Whether o still satisfies q at time t_1 is depicted in column “ *o, q at t_1* ”. Column “*Update Answer Set*” describes which cluster result set has to be updated at time t_1 . Finally, “*Result Updates*” depicts the types of result updates that would be sent, and “*Illustration*” names the figure for that case. For example, in Case 1, o and q are in the same cluster m at both time t_0 and t_1 . At time t_1 o still satisfies q . Since only the updates of the previously reported results are processed, o is neither processed nor is any result sent to the output stream.⁹

⁷ For brevity of discussion, we skip the detailed discussion of each case.

⁸ We did not include the figure for case 6 as it is trivial.

⁹ We omit the detailed discussion of each case, as Table 2 illustrates the main ideas of each case.

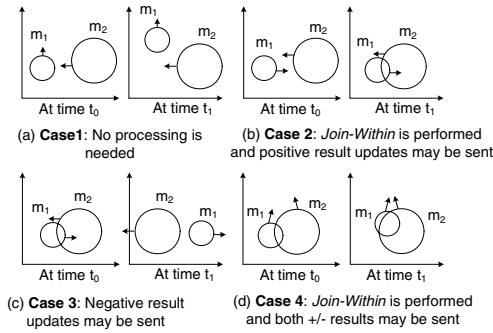


Fig. 6. Join-Between Clusters

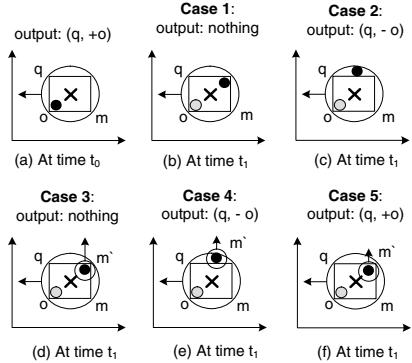


Fig. 7. Join-Within Clusters

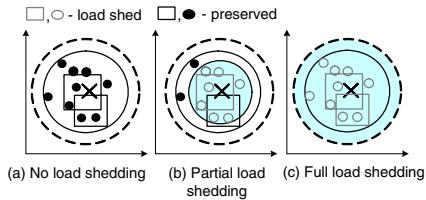


Fig. 8. Effect of Nucleus Size on Load Shedding

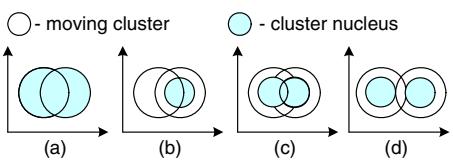


Fig. 9. Affect of Nucleus Size on Join Execution

3 Cluster-Based Load Shedding

Load shedding of moving clusters is an optimization problem composed of three sub-problems: (1) how to effectively estimate the current system load. This implicitly determines when we have to load shed to avoid system overload; (2) how to determine which clusters to load shed. Is it better to shed *all* moving clusters equally or to target a *subset* of moving clusters to more effectively reduce the load, while minimizing the overall inaccuracy; (3) how much to shed per cluster, so that the introduced relative error in the final query results is minimized. We address these three questions next.

3.1 Load Shedding Via Cluster Nucleus

A *nucleus* of a cluster is a circular region that approximates the positions of the members near the centroid of the cluster. Individual positions of objects and queries inside the nucleus are load shed¹⁰. The imprecision is directly proportional to the size of the nucleus. The larger the nucleus, the more imprecise the query results may become.

¹⁰ To load shed a query, the nucleus must fully overlap with the query region.

Definition 3. (*Cluster Nucleus*) Given the nucleus threshold Θ_N where $0 \leq \Theta_N \leq \Theta_D$, let m be a moving cluster. Then the cluster nucleus $m.Nucl$ is the subset of m ($m.Nucl \subseteq m$), where \forall objects $o_i \in m.Nucl$, $|o_i.Loc - m.Loc| \leq \Theta_N$ and \forall queries $q_j \in m.Nucl$, $|q_j.Loc - m.Loc| \leq \Theta_N$.

Fig. 8 depicts the effect of increasing a cluster nucleus on the amount of objects/queries preserved. In Fig. 8a, no load shedding is performed ($\Theta_N = 0$). In Fig. 8b, Θ_N is increased, and seven objects and one query are load shed. Fig. 8c illustrates the extreme case, when $\Theta_N = \Theta_D$ (maximum possible size of the cluster), where all cluster members are discarded. Even if a new member were to arrive to the cluster, it would not be preserved, but automatically discarded. Nucleus threshold Θ_N may be set either globally for all moving clusters, or individually for each moving cluster, if a finer granularity shedding is needed.

Query Processing With Shedded Clusters: By discarding moving entities from the nuclei and not knowing their precise locations, we make several assumptions when executing a cluster-based join. Objects that fall into the intersection region with a nucleus are assumed to satisfy all the queries from the nucleus. Similar reasoning is applied to queries. Fig. 9 depicts the cases with the intersecting clusters and their nuclei.

3.2 Estimating When to Load Shed

Fig. 10 depicts the main steps of load shedding execution. Load shedding is initiated when ClusterShedy utilization becomes greater than or equal to the ρ_{shed} threshold. First, we determine *which* moving clusters are to be shed (termed *shedding clusters*). We then determine *how much per* each shedding cluster to load shed. This is repeated until the current utilization $\rho_{current}$ becomes less than or equal to ρ_{shed_stop} . We use queuing theory [4] and *Little's Law* [13] to predict system overflow¹¹. Queueing theory has been widely used to model and analyze the performance of complex systems involving service. For details of queueing theory see [4][13].

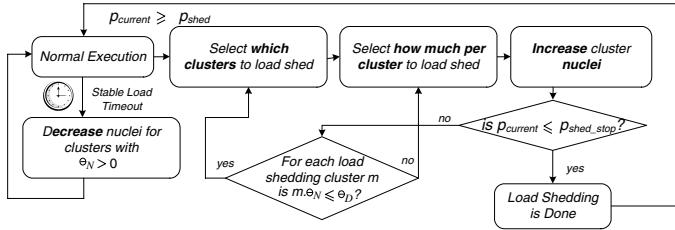
3.3 Estimating Which Clusters to Load Shed

We divide the approaches for picking *which* moving clusters to load shed into two broad categories: *uniform* and *selective*.

Uniform Load Shedding Policies: With uniform load shedding, we load shed across all moving clusters in the system. *Each* moving cluster gets affected by the load shedding procedure, and some or all data within the cluster is discarded. Uniform load shedding policy does *not* mean that all cluster nuclei will increase by *equal* amount. It merely means that all clusters will participate in load shedding and will shed something.

Selective Load Shedding Policies: Alternatively, in selective load shedding, some clusters are chosen for load shedding to minimize the overall inaccuracy

¹¹ However, other models for predicting system overload could be similarly be used.

**Fig. 10.** Load Shedding Execution**Fig. 11.** Score-Based Policy: calculating a score for a moving cluster

of the answers. The different selective policies may include: (1) *Random Policy* – selecting clusters at random; (2) *Count-Based Policy* – selecting clusters with the highest number of members; (3) *Size-Based Policy* – selecting clusters with the smallest size to minimize the overall inaccuracy; (4) *Score-Based Policy* – selecting clusters with the highest scores (see Fig. 11), thus favoring clusters where members are distributed near the centroid regardless of the cluster size; and (5) *Volatility Policy* – selecting clusters with the lowest volatility (i.e., clusters undergoing fewer changes to their properties). The motivation is that stable clusters, once load shed, can accurately approximate their members for longer time intervals, thus amortizing the load shedding overhead in the long term.

3.4 Estimating How Much Per Cluster to Shed

Once clusters have been selected for load shedding, the next question that needs to be addressed is *how much per cluster* to discard, which is determined by the increase in Θ_N . Clusters may either discard all (*total drop*) or only a subset (*partial drop*) of their members.

Total drop policy causes Θ_N to expand to its maximum (Θ_D), and all members inside the clusters are discarded. Total drop is a simple way to quickly reduce the load. However it may significantly impact the accuracy of the results, as we may be discarding data unnecessarily. Partial drop policy is a more prudent way of discarding data. It allows to drop just enough to alleviate the burden on the system without shedding more than necessary.

In the face of detected overload, ClusterSheddy increases cluster nuclei in the *shedding clusters*. The algorithm begins in the exponential growth phase, and increases the nucleus size exponentially¹². Once ClusterSheddy perceives that there is no longer an overload, it starts to *decrease* the nuclei. The rationale for this is that the utilization is below the load shedding threshold, and ClusterSheddy

¹² This is a similar approach as in TCP congestion control mechanisms [15].

could be storing precise locations and producing more accurate query results. It does this by decreasing nuclei a little each time after a periodic timeout, termed *stable load timeout*. For simplicity of presentation, we assume that nuclei radii are decremented by some constant spatial unit amount k each time. Thus ClusterSheddy *multiplicatively increases* cluster nuclei when it detects that utilization is approaching the overload threshold, and *additively decreases* the cluster nuclei when the utilization is low.

4 Experimental Study

Our experiments are based on a real implementation of ClusterSheddy in the Java-based CAPE continuous query engine [23] running on Intel Pentium IV CPU 2.4GHz with 1GB RAM on Windows XP and 1.5.0.06 Java SDK. We use the *Moving Objects Generator* [5] to generate continuously moving objects in the city of Worcester, MA, USA in the form of data streams. We begin with 20K of moving objects and each time unit 1K of new moving objects enter the system. Without loss of generality, all presented experiments are conducted using spatio-temporal range queries. We control skewness of the data and set the *skew factor* to 100. Hence, on average 100 objects are in a cluster. The values of the threshold parameters were set as follows: speed threshold $\Theta_S = 10$ spatial units/time units, distance threshold $\Theta_D = 100$ spatial units, direction threshold $\Theta_R = 10$ degrees, and time threshold $\Theta_T = 1$ time unit. All experimental runs begin with the nucleus threshold set to zero ($\Theta_N = 0$), i.e., no load shedding.

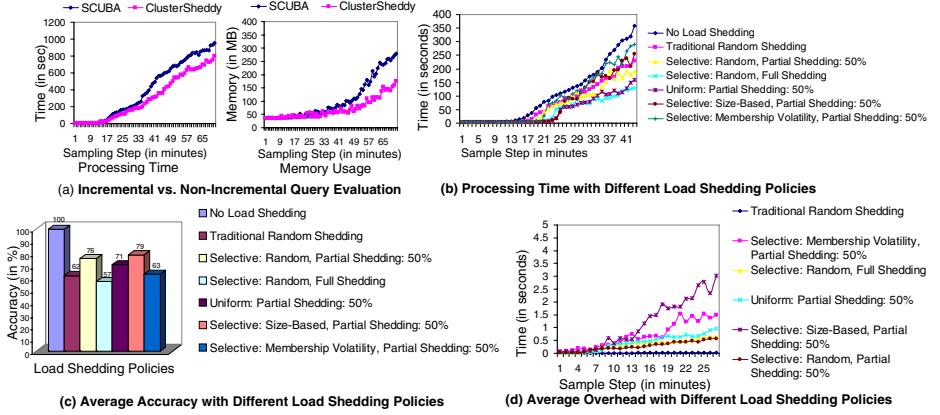
4.1 Incremental vs. Non-incremental Query Evaluation

In this experiment, we compared SCUBA and ClusterSheddy (without load shedding). We wanted to see how the performance and memory consumption are affected when executing spatio-temporal queries using incremental versus non-incremental cluster-based technique (Fig. I2a). We observe that in the long term, ClusterSheddy incremental strategy gives a better performance and requires less memory. The advantage that ClusterSheddy has over SCUBA is that if dense clusters overlap for long durations of time and objects and queries don't change their relative positions within the clusters, the re-evaluation of the contained queries in those clusters is not needed. This translates into significant savings in processing time. Memory-wise, ClusterSheddy also has an advantage over SCUBA, because fewer computations are made and no redundant answers are produced.

4.2 Load Shedding Policies Comparison

We compared different cluster-based load shedding policies¹³ to no load shedding and traditional random load shedding. Figures I2b and I2c respectively represent the join processing times and the accuracy measurements.

¹³ For all full shedding strategies, cluster nucleus is increased to the maximum Θ_D . For all partial shedding strategies, cluster nucleus is increased by 50% each time with respect to the *current* (at the time of shedding) size of the cluster.

**Fig. 12.** Experimental Evaluations

Best Performance: We observe that selective random policy with full load shedding gives the best performance but the lowest average accuracy ($\approx 57\%$). With this policy, clusters are randomly selected and all their cluster members are discarded. The processing overhead is small since clusters are chosen randomly, but the accuracy suffers. Any new objects joining the shedded clusters are assumed to satisfy all queries in the cluster. For any new queries joining these clusters, all cluster objects are returned as satisfying these queries.

Best Accuracy: The best accuracy ($\approx 79\%$) was achieved with the selective size-based partial load shedding policy. Here the smallest clusters were selected first and their nuclei were increased by 50%.

Worst Performance: The worst performance was seen when using selective membership volatility policy with partial shedding. With this policy we picked the clusters that were more stable. However, picking stable clusters did not give much advantage. Very dense clusters may be very dynamic, thus we may not be able to reduce load fast. The membership volatility doesn't account for the count and the distribution of the members within the clusters, hence accuracy may suffer as well.

Overall, if both performance and high accuracy are desired, selective random policy with partial shedding or uniform policy with partial shedding may be used. The former gives a better accuracy ($\approx 76\%$), but has slightly worse performance than the latter policy which has a lower average accuracy ($\approx 71\%$).

4.3 Load Shedding Cost

We compared the load shedding overhead for different policies (Fig. 12d). Load shedding overhead cost includes the time to pre-process the clusters before the shedding is initiated. This may include finding the clusters based on the policy selection criteria (e.g., largest, smallest, most dense, etc.), determining and in-

creasing the nucleus size, associating the ids of the shedded objects and queries with the nucleus, etc. Fig. I2d shows that selective size-based policy has the highest overhead compared to other policies. This is due to the fact that we classify clusters into smaller and larger clusters based on the distribution of their members and also determine if a larger cluster may be an *outlier* cluster¹⁴. Processing the cluster members to see if any latest update caused the cluster to become an “outlier” cluster may require some additional CPU and memory.

5 Related Work

The current state-of-the-art in load shedding includes [1,2,9,22,26,27,28,29]. Load shedding on streaming data has first been presented by Aurora [26,27]. Aurora shedder relies on a quality of service function that specifies the utility of a *tuple*. This is best suited for applications where the probability of receiving each individual tuple in a query result is independent of the other tuples’ values, an assumption that does not hold for spatio-temporal queries.

Load shedding for aggregation queries was addressed in [2]. Babcock et al. describe a load shedding technique based on random sampling. Although sampling works well for aggregation on a traditional data, sampling on location updates without considering their actual values – such as their location – may omit some of the moving entities (when selecting a sample), leading to higher inaccuracy.

The probably most closely related work to ours is the *Scalable On-Line Execution* (SOLE) algorithm [17] performing load shedding on spatio-temporal data streams in PLACE server. In SOLE, specific objects marked as significant are kept, and the other objects are discarded. However, SOLE is not designed to deal accordingly with dense and highly overlapping spatio-temporal data, as objects satisfying many queries are termed as significant and thus are not load shed. ClusterShedy addresses these shortcomings. In fact, it exploits such spatial closeness to approximate the locations when load shedding is performed.

ClusterShedy extends our earlier work – SCUBA algorithm [20], which introduced the concept of moving clusters as abstractions on moving objects. While SCUBA only provides full result recomputation, ClusterShedy now also support incremental query evaluation. Most importantly, ClusterShedy focuses on load shedding – a topic not addressed by SCUBA.

6 Conclusions

This paper addresses an important problem faced in continuous querying of spatio-temporal data streams: system capacity overload. We proposed moving cluster-based load shedding, called ClusterShedy which uses common spatio-temporal properties to determine which objects’ updates would be least sensitive to load shedding and have minimum adverse impact on the accuracy of

¹⁴ We term a cluster an *outlier* cluster, if the majority of its members are distributed near the centroid with an exception of a single member that is at a far distance, thus causing the size of the cluster to increase.

query answers. The proposed technique is general, because moving objects in practice tend to share spatio-temporal properties with other objects for some time intervals. Our experimental results show that *ClusterSheddy* compared to traditional non-spatio-temporal load shedding is efficient in reducing the load while maintaining good accuracy of results.

References

1. B. Babcock, M. Datar, and R. Motwani. Load shedding techniques for data stream systems. In *MPDS: Workshop on Management and Processing of Data Streams*, 2003.
2. B. Babcock, M. Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *ICDE*, pages 350–361, 2004.
3. D. Barbará, W. DuMouchel, and et. al. The new jersey data reduction report. *IEEE Data Eng. Bull.*, 20(4), 1997.
4. G. Bolch and et. al. *Queueing Networks and Markov Chains : Modeling and Performance Evaluation With Computer Science Applications*. John Wiley and Sons, Inc., 1998.
5. T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.
6. D. Carney, U. Çetintemel, and et. al. Monitoring streams - a new class of data management applications. In *VLDB*, pages 215–226, 2002.
7. S. Chu. The influence of urban elements on time-pattern of pedestrian movement. In *The 6th Int. Conf. on Walking in the 21st Cent.*, 2005.
8. A. Das, J. Gehrke, and et. al. Semantic approximation of data stream joins. *IEEE Trans. Knowl. Data Eng.*, 17(1):44–59, 2005.
9. A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD*, pages 40–51, 2003.
10. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.
11. J. A. Hartigan. *Clustering Algorithms*. John Wiley and Sons, 1975.
12. A. K. Jain, M. N. Murthy, and P. J. Flynn. Data clustering: A review. Technical Report MSU-CSE-00-16, Department of Computer Science, Michigan State University, East Lansing, Michigan, August 2000.
13. R. K. Jain. *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, Inc., 1991.
14. P. Kalnis, N. Mamoulis, and et. al. On discovering moving clusters in spatio-temporal data. In *SSTD*, pages 364–381, 2005.
15. J. F. Kurose and K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
16. B. Liu, Y. Zhu, and E. Rundensteiner. Run-time operator state spilling for memory intensive continuous queries. In *SIGMOD Conference*, pages 347–358, 2006.
17. M. F. Mokbel and W. G. Aref. Sole: Scalable online execution of continuous queries on spatio-temporal data streams. tr csd-05-016. Technical report, Purdue University, 2005.
18. M. F. Mokbel, W. G. Aref, and et. al. Towards scalable location-aware services: requirements and research issues. In *GIS*, pages 110–117, 2003.

19. M. F. Mokbel, X. Xiong, and et. al. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *SIGMOD*, pages 623–634, 2004.
20. R. V. Nehme and E. A. Rundensteiner. Scuba: Scalable cluster-based algorithm for evaluating continuous spatio-temporal queries on moving objects. In *EDBT*, pages 1001–1019, 2006.
21. S. Prabhakar and et. al. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Trans. Computers*, 51(10), 2002.
22. F. Reiss and J. M. Hellerstein. Data triage: An adaptive architecture for load shedding in telegraphcq. In *ICDE*, pages 155–156, 2005.
23. E. A. Rundensteiner, L. Ding, and et. al. Cape: Continuous query engine with heterogeneous-grained adaptivity. In *VLDB*, pages 1353–1356, 2004.
24. M. Shah, J. Hellerstein, and et. al. Flux: An adaptive partitioning operator for continuous query systems. cs-02-1205. Technical report, U.C. Berkeley, 2002.
25. A. P. Sistla, O. Wolfson, and et. al. Modeling and querying moving objects. In *ICDE*, pages 422–432, 1997.
26. N. Tatbul. Qos-driven load shedding on data streams. In *XMLDM*, pages 566–576, 2002.
27. N. Tatbul, U. Çetintemel, and et. al. Load shedding in a data stream manager. In *VLDB*, pages 309–320, 2003.
28. N. Tatbul and S. B. Zdonik. Window-aware load shedding for aggregation queries over data streams. In *VLDB*, pages 799–810, 2006.
29. Y.-C. Tu, S. Liu, S. Prabhakar, and B. Yao. Load shedding in stream databases: A control-based approach. In *VLDB*, pages 787–798, 2006.
30. T. Urhan and M. J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Eng. Bull.*, 23(2), 2000.
31. O. Wolfson, H. Cao, and et. al. Management of dynamic location information in domino. In *EDBT*, pages 769–771, 2002.
32. X. Xiong, M. F. Mokbel, and et. al. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE*, pages 643–654, 2005.

Evaluating MAX and MIN over Sliding Windows with Various Size Using the Exemplary Sketch*

Jiakui Zhao, Dongqing Yang, Bin Cui, Lijun Chen, and Jun Gao

School of EECS, Peking University, Beijing 100871, China
`{jkzhao,dqyang,bin.cui,ljchen,gaojun}@pku.edu.cn`

Abstract. MAX and MIN are two important aggregates offered by the original SQL specification. In the paper, we propose a novel mechanism, i.e. the exemplary sketch, to evaluate MAX and MIN over sliding windows with various size in the data stream environment. Performance analysis shows that evaluating MAX or MIN over w sliding windows with various size using the exemplary sketch takes $O(\ln n)$ expected amortized space and $O(w)$ expected amortized evaluation time, where n is the number of the tuples fall into the maximal size sliding window. Moreover, the sliding-window semantics can also be integrated into the exemplary sketch, which means that we no longer need to buffer all the tuples fall into current sliding windows separately for implementing the sliding-window semantics all alone. Experimental results show that the sketch scheme yields very good performance on both space and time cost.

1 Introduction

Recent years, much attention has been focused on online monitoring applications in which continuous queries [1] operate in near real-time over data streams [2] such as web usage logs, network packet traces, etc. Aggregates over sliding windows are an important class of continuous queries for online monitoring over streams and evaluating this class of queries is non-trivial. The potential for high data arrival rates, data bursts, and huge data volumes, along with near real-time requirements of stream applications make space and execution-time performance of stream query evaluation critical. MAX and MIN are two important aggregates offered by the original SQL specification, and are widely used by applications; for example, “*report the highest and the lowest IBM stock price of the most recent one hour, two hours, and four hours in real time*”. In the paper, we introduce how to evaluate MAX and MIN over sliding windows with various size using the exemplary sketch which takes $O(\ln n)$ expected amortized space and $O(w)$ expected amortized evaluation time, where n is the number of the tuples fall into the maximal size sliding window and w is the number of the sliding windows. Moreover, sliding-window semantics can also be integrated into the exemplary sketch, which means that we no longer need to buffer all the tuples fall into current windows separately for implementing the sliding-window semantics.

* Supported by State Key Laboratory of Networking and Switching Technology, NSFC Grant 60473051 and 60503037, and NSFBC Grant 4062018.

1.1 Previous Works

Sliding-window aggregates over streams are an important class of continuous queries, and are the subject of numerous previous works [3], [4], [5]. Recent works about sliding-window aggregates over streams include Jin Li et al. [6], [7]. In [6], the window specification offers two numerical parameters RANGE and SLIDE which specify the window size and how the window slides respectively. Overlapping windows are divided into disjoint “panes”, evaluates sub-aggregates over each pane and “*roll up*” the pane-aggregates to evaluate window aggregates. In [7], the window specification is the same as that in [6]. In order to process each tuple only once, the algorithm maintains aggregate values of each active snapshot window simultaneously and updates aggregate values of each active snapshot window for each incoming tuple. The reason why the algorithms can reduce space and time complexity is that they only maintains “*snapshots*” of query results for each interval of SLIDE long. In the paper, we introduce how to evaluate MAX and MIN over real-time sliding windows with various size by the the exemplary sketch, which has very low space and time complexity and aggregate values are kept up-to-date almost all the time. [9] introduced a similar method, but there is no detailed performance analysis, and the method is “*pull-based*” which means users can only get aggregate values by querying the sketch.

1.2 Paper Outline

The rest of the paper is structured as follows. In Section 2 we introduce the preliminary knowledge of sliding-window aggregates. In Section 3, we introduce how to use the exemplary sketch to evaluate MAX over various sliding windows. Section 4 presents performance analysis of the exemplary sketch. Experimental results are presented in Section 5, followed by our final conclusions in Section 6.

2 Preliminary Knowledge

Tuple-based sliding windows and time-based sliding windows are two important kinds of sliding windows over streams. Suppose $\kappa_{\tau_1} \kappa_{\tau_2} \dots \kappa_{\tau_m}$ are the sequence of tuples of a stream, where τ_i is the timestamp of κ_{τ_i} , $\tau_1 < \tau_2 < \dots < \tau_m$ and κ_{τ_m} is the most recent tuple; if current timestamp is τ_c , semantics of the real-time sliding windows can be characterized as: for a n -tuple sliding window, tuples in $\{\kappa_{\tau_i} \mid m \geq i \geq \max(m - n + 1, 1)\}$ fall into current window; for a s -time sliding window, tuples in $\{\kappa_{\tau_i} \mid (m \geq i) \wedge (\tau_i \geq \tau_c - s)\}$ fall into current window. Sliding-window aggregate queries evaluate aggregates over all the tuples fall into current window, aggregate values change over time as the window slides. For evaluate sliding-window queries, we need to consider how to process arriving tuples and expired tuples as the window slides. Algorithm 1 and Algorithm 2 in Section 3 present how to update the aggregate sketch and evaluate new aggregate values in the case of new tuples arrive at current window and old tuples in current window expire as the window slides. In addition, if the window

Table 1. Taxonomy of aggregates

	COUNT	SUM	AVG	MAX	MIN
<i>combinative</i>	•	•	•	•	•
<i>subtractable</i>	•	•	•		
<i>exemplary</i>				•	•

is a near real-time window, the algorithms for processing arriving tuples and expired tuples must have low time complexity; otherwise, aggregate values can not be kept up-to-date all the time. As shown in Section 2, our algorithms for processing arriving tuples and expired tuples have very low time complexity, so the aggregate values can be kept up-to-date almost all the time even with large numbers of windows. In the rest of the section, we consider why the exemplary sketch should be used to evaluate MAX and MIN over sliding windows; for ease of presentation, we first introduce three dimensions for taxonomy of aggregates.

Combinative. Suppose that an aggregate f over a dataset ξ can be evaluated from a sub-aggregate g over two disjoint datasets ξ_1 and ξ_2 where $\xi_1 \cup \xi_2 = \xi$ and a super-aggregate t , $f(\xi) = t(g(\xi_1), g(\xi_2))$. As defined by Gray et al. [8], if $g = f$, f is *distributive*; if f is non-distributive, but there is a constant bound on the size of the storage needed to store the result of g , f is *algebraic*. Aggregates that are distributive or algebraic are called *combinative* aggregates in the paper.

Subtractable. Suppose there exist two datasets ξ° and ξ , $\xi^\circ \subseteq \xi$. As defined in [9], if there exist two functions g and t satisfying that $f(\xi - \xi^\circ)$ can be evaluated from $g(\xi)$ and $g(\xi^\circ)$, $f(\xi - \xi^\circ) = t(g(\xi), g(\xi^\circ))$, and there is a constant bound on the size of the storage needed to store the result of g , f is *subtractable*.

Exemplary/Summary. As defined in [10], if an aggregate returns one or more representative elements, it is an *exemplary* aggregate; otherwise, it is a *summary* aggregate, for it needs to evaluate some summary properties over all elements.

As shown in Table 1, the five aggregates COUNT, SUM, AVG, MAX, and MIN offered by the original SQL specification are all combinative, and COUNT, SUM, and AVG are subtractable. We can use constant size aggregate sketches to evaluate aggregates that are both combinative and subtractable over sliding windows. For example, an aggregate sketch which records COUNT and SUM can be used to evaluate AVG over sliding windows; when an element arrives at current window, COUNT is increased by 1, and the value of the arriving element is added to SUM; when an element expires, COUNT is decreased by 1, and the value of the expired element is subtracted from SUM. Unfortunately, MAX and MIN are non-subtractable; but they are exemplary aggregates, we may significantly reduce the aggregate sketch size by dropping elements that will not be representative according to aggregate semantics. Accordingly, the aggregate sketch that is used to evaluate MAX and MIN is called the exemplary sketch.

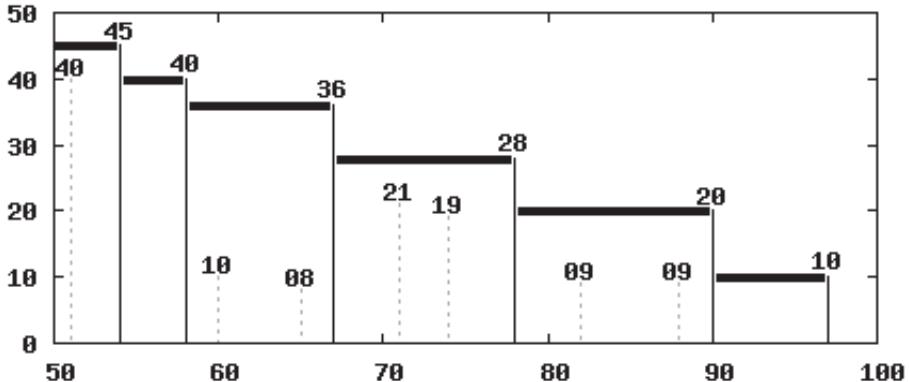


Fig. 1. Future-max elements and non-future-max elements

3 The Exemplary Sketch

In the section, we introduce how to evaluate MAX over real-time sliding windows with various size using the exemplary sketch, the algorithm for evaluating MIN is symmetric. As shown in Section 2 for exemplary aggregates, we may drop elements that will not be representative according to aggregate semantics. An element may be the maximum only if all succeeding elements that have arrived so far are all smaller than it, we call such elements *future-max* elements. Non-future-max elements may be dropped. Fig. 1 shows the sequence of elements of a 50-second sliding window at time 100. Elements are shown using vertical lines with the height indicating the value of the element; future-max elements are shown using solid lines, non-future-max elements are shown using dotted lines.

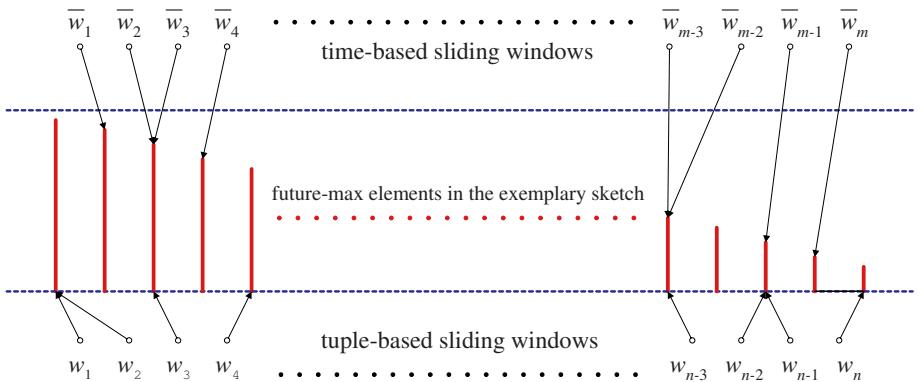
Theorem 1. Future-max elements that ascending ordered by timestamp are descending ordered by value.

Proof. An element can be the future-max only if all succeeding elements that have arrived so far are all smaller than it. For any two future-max elements e_1 and e_2 , if e_2 is an succeeding element of e_1 (the timestamp of e_2 is larger than the timestamp of e_1), the value of e_1 is larger than the value of e_2 . So, future-max elements that ascending ordered by timestamp are descending ordered by value.

We may only save future-max elements in the exemplary sketch. If future-max elements in the exemplary sketch are ordered by timestamp, sliding-window semantics can also be integrated into the exemplary sketch, which means that we no longer need to buffer all the tuples fall into current windows separately for implementing the sliding-window semantics. An exemplary sketch can be seen as a *queue*, an arriving element will be entered into the right hand (each arriving element must first be a future-max element, for there are no succeeding elements that have arrived so far), and future-max elements in the left hand will expire orderly. According to Theorem 1, the left most element in the exemplary sketch

Table 2. Parameters of tuples and sliding windows

	Parameter	Definition
Tuples	<i>value</i>	aggregate attribute value
	<i>position</i>	position in the exemplary sketch
	<i>timestamp</i>	timestamp
	<i>tupleNo</i>	tuple sequence number
Sliding Windows	<i>maxValue</i>	maximal value
	<i>maxValuePosition</i>	position of the maximal value tuple
	<i>minTimestamp</i>	timestamp of the maximal value tuple
	<i>minTupleNo</i>	<i>tupleNo</i> of the maximal value tuple

**Fig. 2.** Evaluating MAX over sliding windows with various size

has the maximal value. An exemplary sketch with the maximal window size can be used to evaluate MAX over sliding windows with various size. Fig. 2 shows the case of evaluating MAX over time-based sliding windows $\bar{w}_1 \bar{w}_2 \dots \bar{w}_m$ and tuple-based sliding windows $w_1 w_2 \dots w_n$. $\bar{w}_1 \bar{w}_2 \dots \bar{w}_m$ and $w_1 w_2 \dots w_n$ are all descending ordered by window size. Each window points to a future-max element in the exemplary sketch, which has the maximal value in the window, and also has the minimal timestamp and minimal tuple sequence number in the window.

Algorithm 1 and Algorithm 2 present how to update the exemplary sketch and evaluate new aggregate values over sliding windows with various sizes in the case of new tuples arrive at current windows and old tuples in current windows expire respectively. Table 2 shows the definitions of the parameters of tuples and sliding windows used by Algorithm 1 and Algorithm 2. The current future-max element will be removed when a new element which is equal or greater than it arrives; so, in Algorithm 1, an arriving tuple will delete elements that are not eligible as a future-max record from the exemplary sketch. Then, the arriving element is entered into the right hand of the exemplary sketch, and revise the

Algorithm 1. Processing an arriving tuple

Input : 1: the arriving tuple γ
 2: the exemplary sketch \hbar
 3: a sequence of tuple-based sliding windows $w_1 w_2 \dots w_n$
 4: a sequence of time-based sliding windows $\overline{w}_1 \overline{w}_2 \dots \overline{w}_m$
 $w_1 w_2 \dots w_n$ and $\overline{w}_1 \overline{w}_2 \dots \overline{w}_m$ are descending ordered by window size

Output: 1: the updated exemplary sketch \hbar
 2: the sequence of updated tuple-based sliding windows $w_1 w_2 \dots w_n$
 3: the sequence of updated time-based sliding windows $\overline{w}_1 \overline{w}_2 \dots \overline{w}_m$

```

1 begin
2   while  $\hbar$  is not empty do
3      $\epsilon \leftarrow$  the right most tuple of  $\hbar$ ;
4     if  $\epsilon.value \leq \gamma.value$  then
5       | delete  $\epsilon$  from  $\hbar$ ;
6     else
7       | break;
8     end
9   end
10  if  $\hbar$  is empty then
11    maxTimestamp  $\leftarrow -\infty$ ;
12    maxTupleNo  $\leftarrow -\infty$ ;
13  else
14     $\epsilon \leftarrow$  the right most tuple of  $\hbar$ ;
15    maxTimestamp  $\leftarrow \epsilon.timestamp$ ;
16    maxTupleNo  $\leftarrow \epsilon(tupleNo)$ ;
17  end
18  enter  $\gamma$  into the right most hand of  $\hbar$ ;
19   $i = n$ ;
20  while  $\gamma(tupleNo) - maxTupleNo \geq w_i.size$  and  $i > 0$  do
21     $w_i.maxValue \leftarrow \gamma.value$ ;
22     $w_i.minTupleNo \leftarrow \gamma(tupleNo)$ ;
23     $w_i.maxValuePosition \leftarrow \gamma.position$ ;
24     $i \leftarrow i - 1$ ;
25  end
26   $i = m$ ;
27  while  $\gamma.timestamp - maxTimestamp \geq \overline{w}_i.size$  and  $i > 0$  do
28     $\overline{w}_i.maxValue \leftarrow \gamma.value$ ;
29     $\overline{w}_i.minTimestamp \leftarrow \gamma.timestamp$ ;
30     $\overline{w}_i.maxValuePosition \leftarrow \gamma.position$ ;
31     $i \leftarrow i - 1$ ;
32  end
33 end

```

parameters of the sliding windows. Algorithm 2 checks whether the future-max element which has the maximal value has expired for each sliding window; if expired, revise the parameters of the sliding window. After that, if the left most element in the exemplary sketch is not pointed by any sliding window, delete it

Algorithm 2: Processing expired future-max tuples

Input : 1: the exemplary sketch \bar{h}
 2: a sequence of tuple-based sliding windows $w_1 w_2 \dots w_n$
 3: a sequence of time-based sliding windows $\bar{w}_1 \bar{w}_2 \dots \bar{w}_m$
 $w_1 w_2 \dots w_n$ and $\bar{w}_1 \bar{w}_2 \dots \bar{w}_m$ are descending ordered by window size

Output: 1: the updated exemplary sketch \bar{h}
 2: the sequence of updated tuple-based sliding windows $w_1 w_2 \dots w_n$
 3: the sequence of updated time-based sliding windows $\bar{w}_1 \bar{w}_2 \dots \bar{w}_m$

```

1 begin
2   if  $\bar{h}$  is empty then return;
3    $\epsilon \leftarrow$  the right most tuple of  $\bar{h}$ ;
4   for  $i=1$  to  $n$  do
5     if  $w_i maxValuePosition$  is not null and
6        $\epsilon.tupleNo - w_i.minTupleNo \geq w_i.size$  then
7        $\xi \leftarrow$  the future-max tuple at position  $w_i maxValuePosition$ ;
8       if  $\xi$  is the right most tuple of  $\bar{h}$  then
9          $w_i maxValue \leftarrow null;$ 
10         $w_i.minTupleNo \leftarrow null;$ 
11         $w_i maxValuePosition \leftarrow null;$ 
12      else
13         $\gamma \leftarrow$  the right next future-max tuple of  $\xi$ ;
14         $w_i maxValue \leftarrow \gamma.value;$ 
15         $w_i.minTupleNo \leftarrow \gamma.tupleNo;$ 
16         $w_i maxValuePosition \leftarrow \gamma.position;$ 
17      end
18    end
19  end
20  for  $i=1$  to  $m$  do
21    if  $\bar{w}_i maxValuePosition$  is not null and
22       $currentTimestamp - \bar{w}_i.minTimestamp \geq \bar{w}_i.size$  then
23       $\xi \leftarrow$  the future-max tuple at position  $\bar{w}_i maxValuePosition$ ;
24      if  $\xi$  is the right most tuple of  $\bar{h}$  then
25         $\bar{w}_i maxValue \leftarrow null;$ 
26         $\bar{w}_i.minTimestamp \leftarrow null;$ 
27         $\bar{w}_i maxValuePosition \leftarrow null;$ 
28      else
29         $\gamma \leftarrow$  the right next future-max tuple of  $\xi$ ;
30         $\bar{w}_i maxValue \leftarrow \gamma.value;$ 
31         $\bar{w}_i.minTimestamp \leftarrow \gamma.timestamp;$ 
32         $\bar{w}_i maxValuePosition \leftarrow \gamma.position;$ 
33      end
34    end
35     $\epsilon \leftarrow$  the left most tuple of  $\bar{h}$ ;
36    if  $\epsilon.position \neq \bar{w}_1 maxValuePosition$  and
37       $\epsilon.position \neq w_1 maxValuePosition$  then
38      | delete  $\epsilon$  from  $\bar{h}$ 
39    end
40  end
41 end

```

from the sketch. In order to keep aggregate values up-to-date all the time, the two algorithms must have very low time complexity. As shown in Section 4, the two algorithms are very time efficient with only a linear expected time complexity.

4 Performance Analysis

In the section, we present our theoretical cost analysis to evaluate the performance of proposed scheme on both space and time. According to the following analysis, we can see that the exemplary sketch only takes $O(\ln n)$ expected amortized space and $O(w)$ expected amortized evaluation time for MIN and MAX aggregate operation, which will be further proved in experimental study.

Theorem 2. Suppose $x_1 x_2 \dots x_n$ are n independent continuous random variables with density function $f(x)$ and distribution function $F(x)$. For each random variable x_i ($1 \leq i \leq n$), if there exists a random variable x_{i° , $i < i^\circ \leq n$ and $x_i \leq x_{i^\circ}$, delete x_i from $x_1 x_2 \dots x_n$. The expected value of the number of the remaining variables, $\mathbb{E}\{NRE\}$ can be characterized by Equation 7

$$\mathbb{E}\{NRE\} \leq 1 + \ln n \quad (1)$$

Proof. For ease of presentation, we define the *zero-one* function $\Phi(i)$ as

$$\Phi(i) = \begin{cases} 1 & \forall t (n \geq t > i) (x_i > x_t) \\ 0 & \text{otherwise} \end{cases} \quad 1 \leq i \leq n.$$

So, iff $\Phi(i) = 1$, x_i is saved. $\mathbb{P}\{\Phi(i) = 1\}$ can be characterized as

$$\mathbb{P}\{\Phi(i) = 1\} = \mathbb{P}\{(x_{i+1} < x_i) \wedge (x_{i+2} < x_i) \wedge \dots \wedge (x_n < x_i)\}.$$

According to probability theory, $\mathbb{P}\{\Phi(i) = 1\}$ can be further characterized as

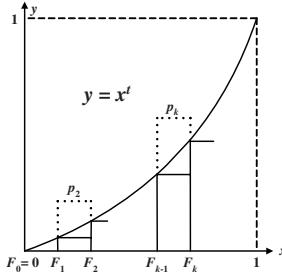
$$\mathbb{P}\{\Phi(i) = 1\} = \int_{-\infty}^{+\infty} \int_{-\infty}^{x_i} \dots \int_{-\infty}^{x_i} f(x_i) f(x_{i+1}) \dots f(x_n) dx_i dx_{i+1} \dots dx_n.$$

According to integral theory, $\mathbb{P}\{\Phi(i) = 1\}$ can be further characterized as

$$\mathbb{P}\{\Phi(i) = 1\} = \int_{-\infty}^{+\infty} f(x_i) F^{n-i}(x_i) dx_i = \int_{-\infty}^{+\infty} F^{n-i}(x) d(F(x)).$$

Replacing $F(x)$ by y , $\mathbb{P}\{\Phi(i) = 1\}$ can be further characterized as

$$\mathbb{P}\{\Phi(i) = 1\} = \int_{-\infty}^{+\infty} F^{n-i}(x) d(F(x)) = \int_0^1 y^{n-i} dy = \frac{1}{n-i+1}.$$

**Fig. 3.** $y = x^t$

The number of the remaining elements $NRE = \Phi(1) + \Phi(2) + \dots + \Phi(n)$, so

$$\mathbb{E}\{NRE\} = \sum_{i=1}^n \mathbb{P}\{\Phi(i) = 1\} = \sum_{i=1}^n \frac{1}{n-i+1} = \sum_{i=1}^n \frac{1}{i}.$$

Since $\ln(n+1) < \sum_{i=1}^n \frac{1}{i} \leq \ln(n) + 1$, $\mathbb{E}\{NRE\}$ can be characterized as

$$\mathbb{E}\{NRE\} = \sum_{i=1}^n \frac{1}{i} \leq 1 + \ln n.$$

Theorem 3. Suppose $x_1 x_2 \dots x_n$ are n independent discrete random variables and $\forall_{x \in x_1 x_2 \dots x_n} (\mathbb{P}\{x = v_k\} = p_k, k = 1, 2, \dots)$. For each random variable x_i ($1 \leq i \leq n$), if there exists a random variable x_{i° , $i < i^\circ \leq n$ and $x_i \leq x_{i^\circ}$, delete x_i from $x_1 x_2 \dots x_n$. The expected value of the number of the remaining variables can also be characterized by Equation \square

Proof. $\Phi(i)$ is the same as defined above, $\mathbb{P}\{\Phi(i) = 1\}$ can be characterized as

$$\mathbb{P}\{\Phi(i) = 1\} = \sum_{k=1}^{\infty} \left(\mathbb{P}\{x_i = v_k\} \cdot \mathbb{P}\{\Phi(i) = 1 \mid x_i = v_k\} \right).$$

According to probability theory, $\mathbb{P}\{\Phi(i) = 1\}$ can be further characterized as

$$\mathbb{P}\{\Phi(i) = 1\} = \sum_{k=1}^{\infty} \left(p_k \cdot \left(\sum_{j=1}^{k-1} p_j \right)^{n-i} \right) = \sum_{k=1}^{\infty} \left(p_k \cdot F_{k-1}^{n-i} \right), \quad F_k = \sum_{j=1}^k p_j.$$

As shown in Fig. 3, the value of $\sum_{k=1}^{\infty} (p_k \cdot F_{k-1}^t)$ can be characterized as

$$\sum_{k=1}^{\infty} \left(p_k \cdot F_{k-1}^t \right) \leq \int_0^1 x^t dx = \frac{1}{t+1}.$$

$\mathbb{P}\{\Phi(i) = 1\}$ can be further characterized as follows, so $\mathbb{E}\{NRE\} \leq 1 + \ln n$.

$$\mathbb{P}\{\Phi(i) = 1\} = \sum_{k=1}^{\infty} \left(p_k \cdot F_{k-1}^{n-i} \right) \leq \frac{1}{n-i+1}.$$

Theorem 4. Algorithm 1 and Algorithm 3 each takes $O(w)$ expected amortized time, where w is the number of the sliding windows.

Proof. In Algorithm 1, for each arriving element, we need to delete the elements that are not greater than the arriving element from the exemplary sketch. Each arriving element will be entered into the exemplary sketch, and will be deleted from the exemplary sketch in the future by Algorithm 1 if it is not greater than a later arriving element or by Algorithm 3 for expiration. On average, Algorithm 1 only needs to delete not greater than one element and takes $O(w)$ expected amortized time. Similarly, Algorithm 3 also takes $O(w)$ expected amortized time.

5 Experimental Results

We have tested the performance of the exemplary sketch over three sets of data generated by the *GNU Scientific Library* on a 1.4 GHz Pentium IV CPU with 2G of memory running RedHat Enterprise Linux Advanced Server. The first set of data is uniformly distributed on [1, 1000], the second and the third set of data are normally distributed with $\sigma=1$ (very skewed data) and $\sigma=2$ (little skewed data) respectively. For each set of data, we change the sliding-window size from 2000 tuples to 20000 tuples stepped by 2000; in each step, we test the space and the time performance of the exemplary sketch during the processing of 20000 arriving tuples. The space performance is measured by the maximal, the average, and the minimal number of the stored future-max elements. For the time performance, Algorithm 3 has already shown that it only checks each sliding window once and has a linear time complexity. On the other hand, Theorem 4 has shown that Algorithm 1 deletes no more than one future-max element on average and has a linear expected time complexity, but the algorithm may delete large numbers of future-max elements and leads to the aggregate values not up-to-date in some rare cases, so we test the maximal and the average number of the future-max elements deleted by each arriving tuple. We have not compared our work with the most recent works [6, 7], for the works have linear space and time complexity for evaluating MAX over only one real-time sliding window.

Fig. 41, Fig. 51, and Fig. 53 characterize the space performance of the exemplary sketch. The average number of the stored future-max elements has a logarithmic growth with the increase of the window size ($\ln 2000 \approx 7.5$ and $\ln 20000 \approx 10$). The maximal number of the stored future-max elements is about double of the average number of the stored future-max elements, and the distribution of the data has very small influence to the number of the stored future-max elements. Fig. 42, Fig. 52, and Fig. 54 characterize the time performance of the exemplary sketch. The average number of the future-max elements deleted

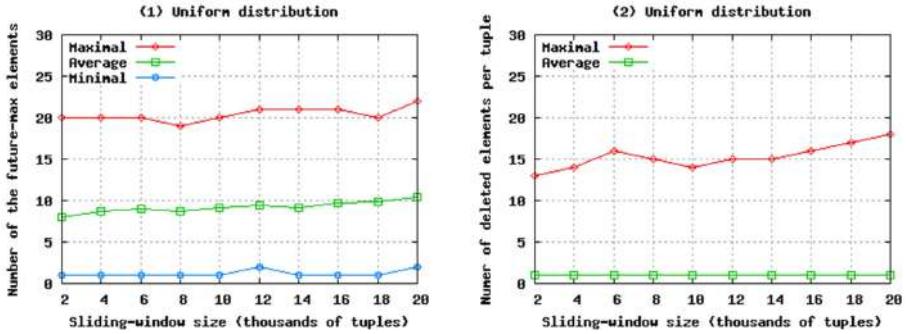


Fig. 4. Performance of the exemplary sketch over elements uniformly distributed on [1, 20000]. The data is tested during the processing of 20000 arriving tuples. (1) The number of the future-max elements stored by the exemplary sketch. (2) The number of the future-max elements deleted by each arriving tuple.

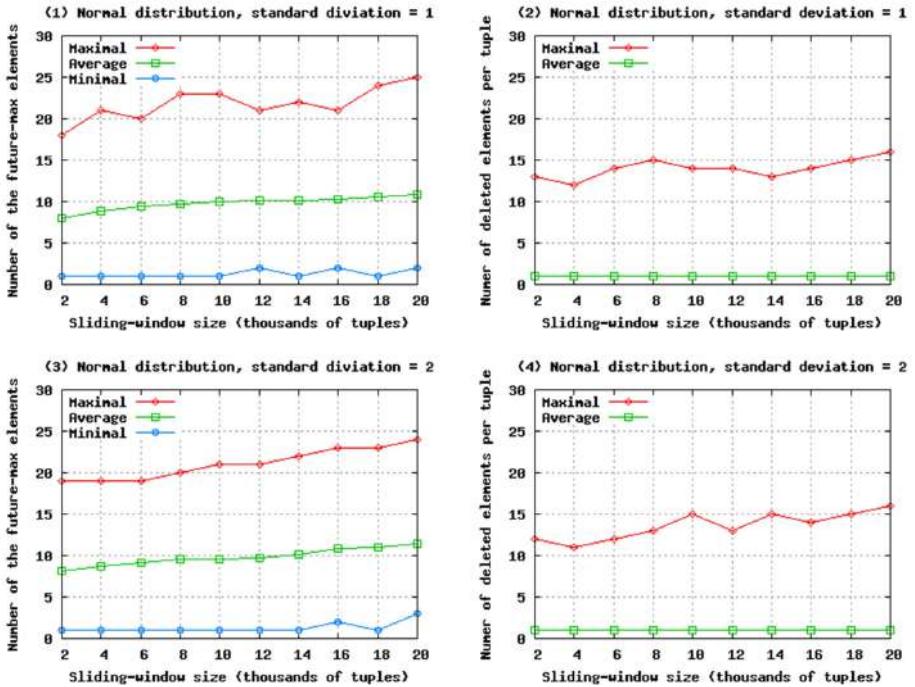


Fig. 5. Performance of the exemplary sketch over normally distributed elements with $\sigma=1$ and $\sigma=2$ resp. The data is tested during the processing of 20000 arriving tuples. (1) The number of the future-max elements stored by the exemplary sketch with $\sigma=1$. (2) The number of the future-max elements deleted by each arriving tuple with $\sigma=1$. (3) The number of the future-max elements stored by the exemplary sketch with $\sigma=2$. (4) The number of the future-max elements deleted by each arriving tuple with $\sigma=2$.

by each arriving tuple is no more than 1. The maximal number of the tuples deleted by each arriving tuple is a little larger than the average number of the stored future-max elements which is very small, so our algorithms can process arriving tuples and expired tuples quickly and keep aggregate values up-to-date almost all the time as long as the number of the sliding-widows is not very large.

6 Conclusions

In the paper, we introduced how to evaluate MAX and MIN over real-time sliding windows with various size using the exemplary sketch which takes $O(\ln n)$ expected amortized space and $O(w)$ expected amortized execution time, where n is the number of the tuples fall into the maximal size sliding window and w is the number of the sliding windows. Moreover, sliding-window semantics can also be integrated into the sketch. So, the exemplary sketch is an extremely good choice for evaluating MAX and MIN over sliding windows in the stream environment.

References

1. Shivnath Babu and Jennifer Widom. Continuous Queries over Data Streams. SIGMOD Record. 2001, 30(3), 109-120.
2. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. Proceedings of PODS 2002, 1-16.
3. Arvind Arasu and Gurmeet Singh Manku. Approximate Counts and Quantiles over Sliding Windows. Proceedings of PODS 2004, 286-296.
4. Sirish Chandrasekaran and Michael J. Franklin. Streaming Queries over Streaming Data. Proceedings of VLDB 2002, 203-214.
5. Phillip B. Gibbons and Srikanta Tirthapura. Distributed Streams Algorithms for Sliding Windows. Proceedings of SPAA 2002, 63-72.
6. Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. No Pane, No Gain: Efficient Evaluation of Sliding-Window Aggregates over Data Streams. SIGMOD Record. 2005, 34(1), 39-44.
7. Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. Semantics and Evaluation Techniques for Window Aggregates in Data Streams. Proceedings of SIGMOD 2005, 311-322.
8. Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals. Data Mining and Knowledge Discovery. 1997, 1(1), 29-53.
9. Arvind Arasu and Jennifer Widom. Resource Sharing in Continuous Sliding-Window Aggregates. Proceedings of VLDB 2004, 336-347.
10. Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a Tiny AGgregation Service For Ad-Hoc Sensor Networks. ACM SIGOPS Operating Systems Review. 2002, 36(SI), 131-146.

CLAIM: An Efficient Method for Relaxed Frequent Closed Itemsets Mining over Stream Data*

Guojie Song^{1,4}, Dongqing Yang¹, Bin Cui¹, Baihua Zheng², Yunfeng Liu³,
and Kunqing Xie⁴

¹ School of Electronic Engineering and Computer Science, Peking University, Beijing, China
gjsong@pku.edu.cn, cuibin@pku.edu.cn, dqyang@pku.edu.cn

² School of Information System, Singapore Management University, Singapore
bhzheng@smu.edu.sg

³ Computer Center of Peking University, Beijing

⁴ National Laboratory on Machine Perception, Peking University, Beijing
kunqing@cis.pku.edu.cn

Abstract. Recently, frequent itemsets mining over data streams attracted much attention. However, mining closed itemsets from data stream has not been well addressed. The main difficulty lies in its high complexity of maintenance aroused by the *exact* model definition of closed itemsets and the dynamic changing of data streams. In data stream scenario, it is sufficient to mining only approximated frequent closed itemsets instead of in full precision. Such a compact but close-enough frequent itemset is called a relaxed frequent closed itemsets.

In this paper, we first introduce the concept of \mathcal{RC} (Relaxed frequent Closed Itemsets), which is the generalized form of approximation. We also propose a novel mechanism CLAIM, which stands for *C*losed *A*pproximated *I*temset *M*ining, to support efficiently mining of \mathcal{RC} . The CLAIM adopts bipartite graph model to store frequent closed itemsets, use Bloom filter based hash function to speed up the update of drifted itemsets, and build a compact HR-tree structure to efficiently maintain the \mathcal{RC} s and support mining process. An experimental study is conducted, and the results demonstrate the effectiveness and efficiency of our approach at handling frequent closed itemsets mining for data stream.

1 Introduction

Recently, data streams emerged as a new data type that attracted great attention from both researchers and practitioners [1]. As a fundamental and essential problem, frequent itemsets mining on data streams has been studied extensively and a large amount of research works have been reported [1][3][4][5][7][8]. Based on our observations, algorithms for mining single item on data stream is efficient enough. However, frequent itemsets mining in such scenario is still confronted with the bottleneck of time and space usage and is still a challenge problem.

The concepts of closed frequent itemsets [1][12] usually can help in accelerating the mining process and compressing the memory usage. However, they can only partly alleviate the problem in data stream, because existing definition of closed itemset require

* This work is supported by the National Natural Science Foundation of China under Grant No. 60473051 and No.60642004 and HP and IBM Joint Research Project.

that the support of closed itemset *exact equals* to the absorbed itemsets, and the slightly support difference aroused by dynamic changing of data stream can lead to the high cost of maintenance. The MOMENT [12] is the first algorithm proposed to mining closed itemsets in data stream scenario, unfortunately it can't solve above problem efficiently since it still follows the traditional definition of closed itemset. In fact, it is unreasonable to have so strict constraints of support exactly equal, because one of the characteristics of data steam mining is a little error tolerant with an approximated mining results.

As introduced in [2], “*most applications will not need precise support information of frequent patterns, a good approximation on support count could be more than adequate. For example, for a frequent itemset {diaper, beer}, instead of giving the exact support count (e.g., 10000), a range, e.g., $10000 \pm 1\%$, may be good enough; the range is a user-specified error bound.*” A condensed FP-base algorithm was also proposed for mining these approximate itemsets, unfortunately it was designed for static data environment, and without giving enough considerations for mining on data stream scenario.

In this paper we propose an approximated frequent closed itemsets model, which we call relaxed frequent closed itemsets (\mathcal{RC}). An efficient algorithm, named Closed Approximate frequent Itemset Mining (CLAIM), is developed by taking advantage of a few interesting techniques, including 1) All frequent relaxed closed itemsets with the same approximated support can be arranged by one bipartite graph model, which can accelerate the drifted itemset update process, 2) To help the drifted itemsets match in bipartite graph, Bloom filter based hash method has been introduced, and 3) A compact tree structure, HR-tree, has been constructed, which combines the above two mechanisms and support the mining process efficiently. We have done extensive experimental study to evaluate the proposed novel algorithm. Experimental results show that CLAIM has significant performance advantage over a representative algorithm for the state-of-the-art approaches.

The remaining of the paper is organized as follows. Section 2 gives the problem statement. In Section 3, we presents the data structure and related techniques of CLAIM, and then introduce the mining algorithm for frequent relaxed closed itemsets based on HR-tree. A performance study of the algorithm is demonstrated in Section 4, and Section 5 discusses the related work. Finally we conclude this paper in Section 6.

2 Problem Definition

Let $I = \{x_1, x_2, \dots, x_n\}$ be a set of items. An itemset is a subset of items I . A data stream, \mathcal{DS} , is a sequence of incoming stream element, (s_1, s_2, \dots, s_N) , where a stream element s_i is an itemset and N is the specified size of the sliding window. The number of stream elements in \mathcal{DS} that contain X is called the support of X , denoted as $sup(X)$. For two itemset X and Y such that $X \subseteq Y$, Y is called a superset of X , and X is a subset of Y . An itemset X is frequent itemset, if and only if $sup(X) \geq s * N$, where $s = sup(X)/N$ is a threshold called minimum support such that $s \in (0, 1)$.

Generally, it is expensive to find the complete set of frequent itemsets, since it is unlikely to own the same support for two itemsets. Moreover, the maintenance cost of closed itemsets will be much higher in data streams because any little concept drift happenings can lead to the changing of closed itemsets. In this paper, to deal with the

approximate frequent closed itemset mining in data stream, we propose the definitions of *relaxed closed itemsets*, denoted as \mathcal{RC} .

Definition 1. Relaxed Interval: The support space of all itemsets can be divided into $n = \lceil 1/\epsilon \rceil$ intervals, where ϵ is user-specified relaxed factor, and each interval can be denoted by $\mathcal{I}_i = [l_i, u_i]$, where $l_i = (n - i) * \epsilon \geq 0$, $u_i = (n - i + 1) * \epsilon \leq 1$ and $i \leq n$.

Definition 2. Relaxed Closed Itemset: An itemset X is called a relaxed closed itemset if and only if there exists no proper superset X' of X such that they belong to the same interval \mathcal{I}_i .

For a relaxed closed itemset X with $sup(X) \in \mathcal{I}_i (= [l_i, u_i])$, if $l_i \geq s$, then X is *frequent* with *frequent interval* \mathcal{I}_i , otherwise X is *infrequent* with *infrequent interval*. For an interval I_i with $l_i < s \leq u_i$, named *critical interval*, it will be divided into two intervals, frequent interval $[l_i, s)$ and infrequent interval $[s, u_i)$.

Example 1. Table 1 shows a sample stream data \mathcal{DS} , where ϵ is 20%, s is 45% and the sliding window size N is 6, from tid=1 to tid=6.

Table 1. An example of stream dataset

Transaction Id	Stream Elements
1	a, b, c, d, e
2	c, d, e
3	a, b, c, e
4	a, c, d, e
5	a, b, c, d, e
6	b, c, d
7	b, c, e
8	...

With the support of definition 1 and 2, relaxed itemsets can be generated based on dataset table 1, as shown in Table 2

Table 2. Relaxed closed itemsets

\mathcal{I}_i	Itemsets	$\mathcal{RC}_{\mathcal{I}_i}$
$\mathcal{I}_1 = (0.8, 1]$	c,d,e,cd,ce	cd,ce
$\mathcal{I}_2 = (0.6, 0.8]$	a,b,ac,ae,bc,de,ace,cde	bc,ace,cde
$\mathcal{I}_c = [0.45, 0.6]$	ab,ad,be,abc,abe,acd,ade,bce,abce,acde	abce,acde
$\mathcal{I}_n = [0, 0.45]$	abd,bde,abcd,abde,bcde	abcde

Above relaxed closed itemset model has absorbed more frequent itemsets than traditional one under the constraint of relaxed factor, achieved a good compression effect. Interestingly, traditional closed itemset mining can be achieved with the setting of $\epsilon = 1/N$, while the maximal frequent itemsets mining has the same effect with our method if we set ϵ to 1. Thus, our model is a generalized formal of above two extreme cases.

Our task is to mine frequent relaxed closed itemsets with relaxed factor ϵ on N -size stream sliding window.

3 CLAIM: Mining Relaxed Frequent Itemsets on Data Streams

In this section, we perform step-by-step analysis to develop an efficient scheme, CLAIM, for relaxed frequent closed itemsets mining.

3.1 Bipartite Graph Model

Dynamic changing of data distribution is an intrinsic characteristic of data stream, which can lead to the adjustment of \mathcal{RC} frequently. In such case, only maintaining this upper bound can not track the bound drift efficiently. Similar with upper bound definition, *lower bound* can be defined for each interval.

Example 2. Based on table 2, the double bound of relaxed closed itemsets can be illustrated as Table 3.

Table 3. An example of double bound

\mathcal{I}_i	Lower bound	Upper bound
$\mathcal{I}_1 = (0.8, 1]$	c,d,e	cd,ce
$\mathcal{I}_2 = (0.6, 0.8]$	a,b,de	bc,ace,cde
$\mathcal{I}_c = [0.45, 0.6]$	ab,ad,be	abce,acde
$\mathcal{I}_n = [0, 0.45)$	abd,bde	abcde

At present, all itemsets within double bound are disordered, which is unfavorable for efficient update of \mathcal{RC} . For example, if itemset x in lower bound has been deleted from current interval, its upper pattern x' should be generated as the substituted itemset in lower bound by scanning each itemset y in upper bound satisfying $x' \subseteq y$. However, the cost of such itemset maintenance operation with the support of such disordered double bound is $O(n)$, where n is the number of itemsets in upper bound. Thus, to reduce the cost of update, a data structure, *bipartite graph* has been introduced here.

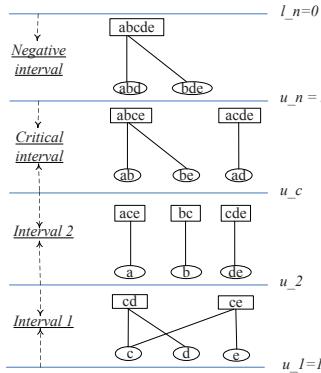
Definition 3. Bipartite Graph: A bipartite graph $BG = (U, L, E)$ has two distinct vertex sets U and L with $U \cap L = \emptyset$, and edge set $E = \{(u, l) | u \in U \wedge l \in L\}$.

In our case, for an interval \mathcal{I}_i , U_i (L_i) is a subset of upper bound (lower bound). The edge $e(u, l) \in E$ ($u \in U_i$, $l \in L_i$) means that there exists *including* relationship between itemset u and v , $u \supset v$. We also know that all absorbed itemsets can be deduced in each bipartite graph.

Lemma 1. Bipartite Graph Decomposition: A bipartite graph $BG = (U, L, E)$ can be decomposed into n independent bipartite graph BG_1, \dots, BG_n , if and only if $U_i \cap U_j = \emptyset$, $L_i \cap L_j = \emptyset$ and $E_i \cap E_j = \emptyset$ for any $i, j \in [1, n]$.

Example 3. Continuing with example 1, all these itemsets in double bound can be decomposed into a set of bipartite graphs, as shown in Figure 1. For example, bipartite graph $BG = (\{abde, acde\}, \{ab, ad, be\}, \{e(abde, ab), e(abde, be), e(acde, ad)\})$ in interval 3 can be decomposed into two sub-bipartite graphs $BG_1 = (\{abde\}, \{ab, be\}, \{e(abde, ab), e(abde, be)\})$ and $BG_2 = (\{acde\}, \{ad\}, \{e(acde, ad)\})$.

By using bipartite graph model, the cost of itemset update can be reduced to $O(m)$, where m is the number of itemset in upper(lower) bound of bipartite with $m \ll n$.

**Fig. 1.** Bipartite graph model

3.2 Bloom Filter Based Hash Function

As concept drift happening in data stream, the \mathcal{RC} s within double bound must be maintained. For each *drifted itemset* X (if $sup(X) < l_i$ or $sup(X) \geq u_i$ in interval \mathcal{I}_i), it should be deleted from original bipartite graph and inserted into target bipartite graph in its adjacent interval \mathcal{I}_{i+1} (or \mathcal{I}_{i-1}). The core problem is how to find such target bipartite graph(s) from a large amount of candidates, named *drifted itemset location*.

To complete such location process, a straightforward method is needed to match all itemsets in all bipartite graphs within interval \mathcal{I}_{i+1} (or \mathcal{I}_{i-1}) to check which bipartite graph it belongs to. However, such process is very time-consuming. An ideal method is to find the destination bipartite graphs directly without checking irrelevant itemsets. To achieve this, we adopt *Bloom filters* [13] based hash method to help find its target.

Suppose all items in itemsets of bipartite graphs are sorted according to specified sequence, such as natural order, $I = \{a, b, \dots, y, z\}$ in our example. we have following hash function definition.

Definition 4. Hash function: For any item $x \in I$ and itemset X , if $x \in X$, then the position of x located in I is set to 1 else 0. In such case, each itemset will be correspond to one bit sequence composed of 0 and 1, which also can be expressed in one integer number, denoted as

$$\mathcal{F}(X, I) = \sum_{x \in X} 2^{(pos(x)-1)} \quad (1)$$

where $pos(x)$ refers to the position of item x in I in right-first order. Thus our hash function can be defined as follow:

$$\mathcal{H}(X) = \{h_i(X) | h_i(X) = mod(\mathcal{F}(X, I)^i, m)\} \quad (2)$$

where $0 < i \leq K$.

However, above hash function can not be used directly in our case. The reason is that Bloom Filter based hash method only supports membership query, that is, only tell us whether an itemset X belongs to hash set BF or not. Unfortunately, drifted itemset

match needs the support of *include* or *contain* relationship operation between itemsets. To apply this technique, we define two sets, BF_u and BF_l , for each bipartite graph BG .

Definition 5. BF_u : For any bipartite graph $BG = (U, L, E)$, its BF_u is composed of itemset X , $X \notin BG$, which is contained by at least one itemset Y in U , but not exist any $X' \notin BG$ with $X' \subset Y$ s.t. $X' \supset X$.

The generated process of BF_u can described as follows: for each itemset X of length k in U , all its subsets with length $(k-1)$, denoted as S_{k-1} , can be enumerated. For each $X' \in S_{k-1}$, it will be filtered out if it contains an itemset in L or it is contained by an itemset in BF_u . Otherwise, it can be inserted in BF_u . For each filtered itemset, continuing such process until all itemsets in BF_u have been generated.

With the guidance of above method, for bipartite graph $BG(= \{\{abce\}, \{ab, be\}, \{e(abce, ab), e(abce, be)\}\})$ in Figure 1, its BF_u can be deduced with $\{ace, bc\}$.

Lemma 2. An upper bound drifted itemset X is contained by BG , if and only if we have $X \in BF_u$.

Proof. Assume there exists an upper bound drifted itemset X , if it is contained by an itemset in U of BG , but with $X \notin BG$, then we must have $X \in BF_u$, otherwise there should be itemset $X' \in BF_u$, $X' \supset X$, which means X' is also an upper bound drifted itemsets, which is contrary with the definition of drifted itemset. If $X \in BF_u$, then X must be absorbed by one itemset in U of BG . Thus we have lemma.

Definition 6. BF_l : For any bipartite graph $BG = (U, L, E)$, its BF_l is composed of itemset X , $X \notin BG$, which contains at least one itemset Y in L , but not exist any $X' \notin BG$ with $X' \supset Y$ s.t. $X' \subset X$.

The generation of BF_l can be simulated with that of BF_u . In Figure 1, the BF_l of bipartite graph $BG(= \{\{ace\}, \{a\}, \{e(ace, a)\}\})$ can be deduced with $\{ab, ad\}$.

Lemma 3. A lower bound drifted itemset X is contained by BG , if and only if we have $X \in BF_l$.

Proof. Similar with lemma 2, omitted here.

Thus, each bipartite graph corresponds to two hash set BF_u and BF_l . Drifted itemset locating can be finished in $O(1)$ time by using the Hash function in equation (2).

3.3 Structure of HR-Tree

With the consideration of time and space limitation under data stream scenario, all bipartite graphs in our model have been arranged into one compact prefix tree structure, named **HR-tree** (**H**ash based **R**elaxed Closed Itemset tree), which not only can saving storage space and reducing the cost of itemset counting with the update of sliding window, new stream element insertion and old one deletion, but also combine the Bloom filter based hash technique introduced in section 3.2.

As illustrated in Figure 2, HR-tree is composed of two parts: *hash table* and *prefix tree*. In hash table, each interval corresponds to a set of bipartite graphs and each bipartite graph has its two hash function entries: $\mathcal{H}_i(BF_u, X)$ for upper bound and

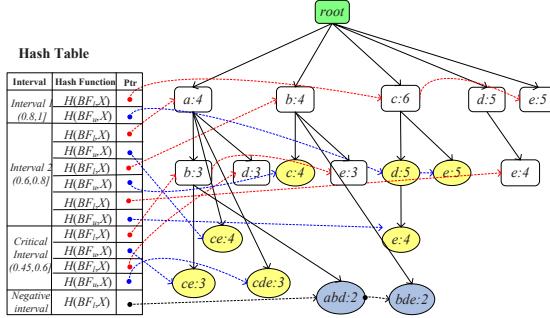


Fig. 2. HR-tree

$\mathcal{H}_i(BF_l, X)$ for lower bound. Pointer ptr points to the itemset list in upper bound or lower bound of BG , ending with tag *null*. All itemsets in double bounds of bipartite graphs will be arranged by prefix order. Each node corresponds to one itemset with its support sup . We will not present the detail construction algorithm as the basic structure is same as the well known Prefix-tree [9].

For any given drifted itemset X , the decision whether it belongs to bipartite graph BG or not can be answered directly with the answering of hash function entry $\mathcal{H}_i(BF_u, X)$ or $\mathcal{H}_i(BF_l, X)$, no need to access all related itemsets in *prefix tree*.

3.4 The CLAIM Algorithm for \mathcal{RC} Mining

For the maintenance of a bipartite graph, only its upper bound drift is cared as the insertion of stream element, and only its lower bound drift is cared as the deletion of stream element. These two operations correspond to the similar maintenance techniques, thus only the pseudo algorithm of insertion operation is presented in Figure 3.

Initially, HR-tree is *null* and relaxed interval is divided firstly according to definition 2.1. There are no \mathcal{RC} s generated at the beginning, thus they should be initialized by the insertion of stream element continuously until sliding window is full(in line 1).

For each incoming stream element, support counting is executed by scanning HR-tree starting from *root* in prefix-order, and then drifted itemsets can be collected directly. However, the drifted itemsets contained by bipartite graph but not emerged in double bound, named *internal drifted itemsets*, can not be found immediately, since no support information has been recorded. Thus, for each drifted itemset X , the contained internal drifted itemsets must be counted firstly by scanning all stream elements within sliding window, and are checked whether they also belong to drifted itemsets (in line 3-9).

Thereafter, all (internal) drifted itemsets will be deleted firstly from original bipartite graph, and it can be decomposed into several independent bipartite graphs possibly. By using drifted itemset location operation with the help of hash function, bipartite graphs containing drifted itemset should be combined simultaneously. The decomposition and combination of bipartite graph can be maintained easily by adjusting *side-link* pointed by pointer ptr (in line 10-16).

Time Complexity includes: 1) *Support counting* based on HR-tree is efficient because it can be finished in prefix order and no need scan all itemsets; 2) *Drifted itemsets location*

```

Input:  $DS$ : data stream in sliding window,  $\epsilon$ : relaxed threshold;
 $s$ : the minimum support threshold;

1.  $HR\text{-tree} = null$ ,  $\mathcal{I} = \bigcup_0^{\lceil 1/\epsilon \rceil - 1} \mathcal{I}_i$ ,  $\mathcal{RC} = \text{initialize}(DS)$ ;
2. for each incoming stream element  $e \in DS$  do
3.    $drifted\text{-itemset} = \text{scan}(HR\text{-tree}, e)$ ;
4.   for  $X \in drifted\text{-itemset}$  belonging to interval  $i$  do
    /* find all internal drifted itemsets contained by  $X$  in  $BG$  */
5.      $BoundSet = X$ ;
6.     for each sub-itemsets  $x \subset X$  in  $BG$  do
7.        $sup(x) = \text{explore}(x, DS)$ ;
8.       if ( $sup(x) \equiv sup(X)$ ) and ( $\exists p \in BoundSet$  st.  $x \in p$ ) then
9.          $BoundSet = (BoundSet - p)$ ,  $BoundSet = BoundSet \cup x$ ;
    /* bipartite graph decomposition */
10.     $BG = \text{Removed drifted itemsets in } BG$ ;
11.    if ( $BG_1^i \cap \dots \cap BG_m^i \equiv null$ ) then /*
12.       $BG$  is decomposed into  $BG_1^i, \dots, BG_m^i$  with HR-tree adjustment;
    /* bipartite graph combination */
13.    for each  $x' \in BoundSet$  do
14.       $\{BG_1^{i-1}, \dots, BG_n^{i-1}\} = \mathcal{H}_{i-1}(BF_i, x')$ ; /* drifted itemset locating */
15.      if ( $n > 1$ ) then
16.         $BG_{new} = \text{combination}(BG_1^{i-1}, \dots, BG_n^{i-1}, x')$  with HR-tree adjustment;
17. for each leaving stream element  $e \in DS$  do {...} // We omit the details here.

```

Fig. 3. Algorithm CLAIM(DS, ϵ, s)

can be finished in $O(1)$ with the help of hash function; 3) The cost of *Internal drifted itemset counting* is $O(\alpha * \mathcal{N})$, where α is the number of internal drifted itemsets, and \mathcal{N} is sliding window size. Although such cost is a little higher, α is always small especially when dynamic changing of data distribution is not so big. Moreover, some optimization techniques based on *apriori* property has been developed to reduce the number of α , omitted here for space constraint.

Space Complexity includes two aspects: *HR-tree* and *Hash space*. 1) The former is a compact prefix structure, which can save the memory usage as little as possible. In the worst case ($\epsilon = 1/\mathcal{N}$), memory usage is identical with MOMENT ($U=L$); 2) Since Bloom filter based hash space is composed of δ bit, thus the memory usage is $O(\delta * n)$, where n is the number of bipartite graph. Moreover, the support accuracy of mining results can be controlled by adjusting our relaxed threshold ϵ according to the memory constraints, because there is a tradeoff between accuracy and memory usage.

4 Experimental Evaluation

To evaluate the effectiveness and efficiency of our algorithm CLAIM, we conduct a comprehensive set of experiments. We implement MOMENT [12] as the baseline algorithm to generate closed frequent itemsets. All the experiments are run on the PC with Intel Pentium R CPU 1.5 GHz, 1G MB memory, and OS Microsoft XP. All the programs are written in Microsoft/Visual C++6.0.

We generate transactional data stream using IBM data generator described by Agrawal et al, which mimics the data stream of retailing. We generate one type of data streams, T10I4D100K, where 10 and 4 are the average size of a transaction and a maximal frequent itemsets of the two streams, respectively, and the default size of the sliding window \mathcal{N} is 80,000. We generate 81,000 transactions and we report the average performance over 1000 consecutive sliding windows (each with size 80,000). Four tuning parameters have been studied in our experiments: relaxed interval ϵ , minimum support threshold s , sliding window size \mathcal{N} , and data arrival order.

4.1 Effect of Minimum Support

The influence of minimum support threshold s for running time and memory usage has been evaluated in this section. Figure 4 shows the running time and memory usage of the two algorithms with $\epsilon = 0.1\%$, $\mathcal{N} = 8 * 10^4$, and s ranging from 0.01% to 1%. Figure 4(a) show that CLAIM can achieve better running time than MOMENT. For example, when minimum support s is set to 0.4%, the running time of MOMENT is 4.9 while that of CLAIM is 1.9. The reason is that there are less drifted itemsets with less support counting for CLAIM than MOMENT. Figure 4(b) shows the similar case for memory usage. However, when s start to less than 0.2%, the memory usage begin to improved more quickly for MOMENT than CLAIM, which shows that our relaxed interval can condense frequent itemsets efficiently.

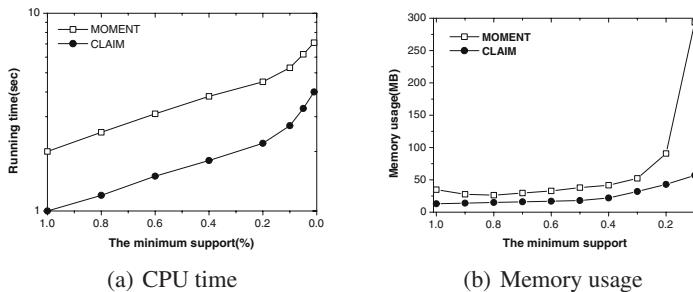


Fig. 4. Varying s ($\epsilon = 0.1\%$, $\mathcal{N} = 8 * 10^4$)

4.2 Effect of Relaxation Interval

Figure 5 shows the running time and memory usage of our algorithm with varying the relaxation interval ϵ . We set $s = 0.4\%$ and $\mathcal{N} = 8 * 10^4$. For CPU time, with the increasing of relaxed interval threshold, the running time decreases obviously, because only itemsets in double bound need to be checked and maintained, and the concept drift of internal patterns contained have been omitted. Especially, when ϵ becomes $1/\mathcal{N}$, the functions of these two algorithms are identical, and the mining results are also the same. At this time, CPU time(4 sec) of MOMENT is slightly higher than that of CLAIM(4.5 sec), which should due to our hash method. The larger ϵ is, the less the cost of maintaining process. For memory usage, it shares the similar results as CPU time. The different case is that when ϵ becomes $1/\mathcal{N}$, the memory usage of MOMENT(50MB) is less than CLAIM(60MB), which should be invoked by our hash table.

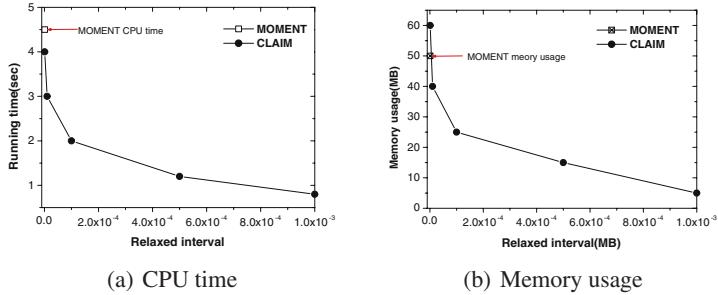


Fig. 5. Varying ϵ ($s = 0.4\%$, $N = 8 * 10^4$)

4.3 Effect of Sliding Window Size

We test the impact of the sliding window size N on CPU time and memory usage of two algorithms. We fix $\epsilon = 0.1\%$ (for CLAIM only), $s=0.4\%$, and vary the sliding window size N from $6 * 10^4$ to $1 * 10^5$. The CPU time and memory usage are shown in Figure 6. CLAIM significantly outperform MOMENT. When dealing with a $8 * 10^4$ data stream, MOMENT requires 4.8 seconds to process it, while as CLAIM need only about its 46%, 2.2 seconds. Also, MOMENT consume 53MB memory, while CLAIM only needs 22MB.

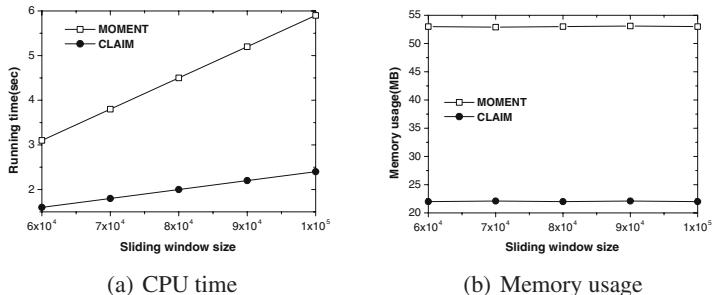
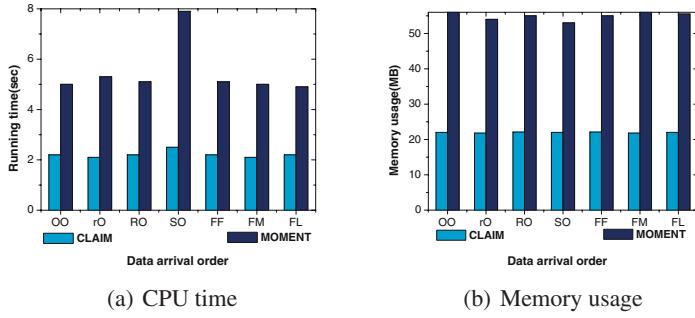


Fig. 6. Varying N ($\epsilon = 0.1\%$, $s = 0.4\%$)

4.4 Effect of Data Arriving Order

Similar as method adopted in [1], we test data arrival orders, in order to ensure whether our approach is order sensitive. Let $s=0.4\%$, $\epsilon = 0.1\%$, and $N = 8 * 10^4$. Several data arriving orders are tested: *OO*(Original Order), *rO*(Reverse Order), *RO*(Random Order), *SO*(segment-based random order)¹, *FF*(Frequent First), *FM*(Frequent Middle) and *FL*(Frequent Last). As shown in Figure 7, MOMENT is more sensitive to data arriving orders regarding to frequent closed itemsets mining than CLAIM, especially when arriving order *SO* is has been chosen. For insertion, CLAIM outperforms MOMENT in terms of CPU time and memory consumption.

¹ We randomly reordered data in a unit of segment(1000 items) from data set T10I4D100K and T20I6D100K.

**Fig. 7.** Varying \mathcal{N} ($\epsilon = 0.1\%$, $s = 0.4\%$)

5 Related Works

Recently discovering frequent itemsets has been successfully extended to data stream mining, which is more challenging than mining in transaction databases. Manku et al [5] gave an algorithm called LOSSY COUNTING for mining all frequent itemsets over the entire history of the streaming data. Giannella et al [6] proposed an approximate algorithm for mining frequent itemsets in data streams during arbitrary time intervals. An in-memory data structure, FP-stream, is used to store and update historic information about frequent itemsets and their frequency over time and an aging function is used to update the entries so that more recent entries are weighted more. Jeffrey et al [11] proposed one method for false-negative oriented frequent itemsets mining, where the number of false-negative itemsets can be controlled by a predefined parameter so that desired recall rate of frequent itemsets can be guaranteed. Chang et al [3] developed an algorithm for maintaining frequent itemsets in streaming data assuming each transaction has a weight that is related with its age. Besides, Charikar et al [4], Cormode et al [7] and Richard et al [8] also proposed algorithms for efficient single item mining.

In all above studies, frequent itemsets mining algorithms are based on the Apriori property, which try to use filter and fast counting technique to support approximate frequent itemsets mining. In spite of this, they can not overcome the efficiency problem once the scale of frequent itemsets is very large. The large number of frequent itemsets makes it impractical to maintain information about all frequent itemsets using in-memory data structures. Chi et al [12] proposed an algorithm called MOMENT to mining all closed itemsets exactly from data streams with the support of traditional closed frequent itemset definition. In contrast, our algorithm is a flexible one because we relaxed the definition of closed itemset that allow a gap (not exact the same) between closed itemset and absorbed itemsets. In this way, the mining of the closed itemset will be much flexible which can be controlled by users with different accuracy requirements.

6 Conclusion

We have studied a practically interesting problem, mining relaxed frequent closed itemsets, a general form of traditional closed itemsets mining, and proposed an efficient

algorithm, CLAIM, with following contributions: (1) giving out a more general definition of approximate closed itemsets mining, (2) introducing bipartite graph model to alleviate itemsets update cost, (3) using a Bloom filter based hash function to quicken drifted itemsets search, and (4) proposing a compact structure–HR-tree to help maintain relaxed frequent closed itemsets efficiently. To the best of our knowledge, this is the first research result for mining approximate frequent itemset in data stream scenario.

Based on this study, we conclude that mining relaxed frequent closed patterns on data stream environments with adjustable relaxed factor ϵ should be more preferable than the traditional *exact-support-equal* based mining for frequent closed itemsets. More detailed study along this direction is needed, including dealing with a stream segment each time not only one element, as well as mining relaxed frequent closed sequential patterns.

References

1. Jeffrey Xu Yu, Zhihong Chong, Hongjun Lu, Aoying Zhou. False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams. In Proc. of the 28th Intl. Conf. on Very Large Data Bases, pages 204-215, 2004.
2. Jian Pei, Guozhu Dong, Wei Zou, Jiawei Han. On Computing Condensed Frequent Pattern Bases. In Proc. of IEEE Intl. Conf. on Data Mining, pages 378-385, 2002.
3. J. H. Chang and W. S. Lee. Finding recent frequent itemsets adaptively over online data streams. In Proc. of the 9th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, pages 487-492, 2003.
4. M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In Proc. of the 29th Intl Colloquium on Automata, Languages and Programming, 2002.
5. G. Manku and R. Motwani. Approximate frequency counts over data streams. In Proc. of the 28th Intl. Conf. on Very Large Data Bases, pages 346-357, 2002.
6. C. Giannella, J. Han, E. Robertson, and C. Liu. Mining frequent itemsets over arbitrary time intervals in data streams. Technical Report tr587, Indiana University, 2003.
7. G. Cormode, S. Muthukrishnan, What's Hot and What's Not: Tracking Most Frequent Items Dynamically, In the ACM Symposium on Principles of Database Systems, pages 296-306, 2003.
8. Richard M. Karp, Scott Shenker, A Simple Algorithm for Finding Frequent Elements in Streams and Bags, In the ACM Transactions on Database Systems, 28(1):51-55, 2003.
9. J. Han, J. Pei, and Y. Yin. Mining frequent itemsets without candidate generation. In Proc. of the ACM SIGMOD Intl. Conf. on Management of Database, pages 1-12, 2000.
10. K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In Proc. of the 2001 IEEE Intl. Conf. on Data Mining, pages 163-170, 2001.
11. J. Wang, J. Han, and J. Pei. Closet+: searching for the best strategies for mining frequent closed itemsets. In Proc. of the Intl. Conf. Knowledge Discovery and Data Mining, pages 236-245, 2003.
12. Y. Chi, H. Wang, P. Yu, and R. Muntz, MOMENT: Maintaining closed frequent itemsets over a stream sliding window. In Proc. Of 4th IEEE Intl. Conf. on Data Mining, pages 59-66, 2004.
13. B. Bloom. Space/time tradeoffs in in hash coding with allowable errors. Communications of the ACM, 13(7):422-426, 1970.

Capture Inference Attacks for K-Anonymity with Privacy Inference Logic

Xiaojun Ye¹, Zude Li², and Yongnian Li¹

¹ School of Software, Tsinghua University, Beijing 100084, China

yexj@tsinghua.edu.cn, liyongnian04@mails.tsinghua.edu.cn

² Computer Science Department, University of Western Ontario, Canada
zli263@uwo.ca

Abstract. General k-anonymity models cannot fully prevent individual re-identification on released microdata since intruders can capture various prior information to recognized individual identities with enough precision, ultimately, to precisely associate these identities with some sensitive attribute values. We propose the privacy inference logic to specify the k-anonymity method and emphasize the nature of privacy inference attacks on k-anonymized microdata in a more rigorous way (than the previous work, including ℓ -diversity), which can be used as a guideline and a predictor for efficient privacy disclosure control during k-anonymization process on a pre-published microdata set. Based on this theory, we uncover and define several main inference attacks classes in aspect of the various “knowledge” used by intruders. In experiments, we successfully implement the probabilistic inference risks evaluation as factors considered in an anonymization cost metric as a more effective anti-inference k-anonymity solution.

1 Introduction

An anonymized individual represented as a record in the released database might be recognized by an intruder through the data with external identification database or with individuals from his circle of acquaintances. To reduce the risk of this attack type, *k-anonymity* is proposed as a mechanism for privacy protection against individual re-identification in microdata publishing [14][17]. Many instances illustrating such attacks are listed in literature such as [7][8][12][17][18][20][21], as the motivations for most *k-anonymity* models introduced in the past several years. In general, *k-anonymity* means that one can only be certain that a value is associated with one of at least k values, or in a *k-anonymized* dataset, each record is indistinguishable from at least $k-1$ other records with respect to (w.r.t.) certain identifying attributes [17]. A *privacy inference attack* is roughly defined as to find some private information existed in the original microdata that is concealed literally in the k-anonymized microdata, such as re-identification disclosure or prediction (attribute) disclosure discussed in [7].

Current k-anonymity models are not robust enough to handle many kinds of privacy inference attacks. But the most unfortunate is that such attacks have not been fully cognized or have been ignored when building and using these

models. The objective for most k-anonymity models is to disseminate statistical microdata in such a way that individual privacy is sufficiently protected against recognition of the subjects to which it refers, which at the same time providing society with as much as possible under this k-anonymity dataset.

For example, Table 1 shows a set of history medical records from a fictitious hospital. We firstly remove the Name (direct identifier) attribute column and then use a 2-anonymity model on the table with data generalization approach [16] and get the 2-anonymized table (Table 2). Here we take * to denote a generalization value and $<_{\nu}$ to indicate a value generalization relation on an attribute. For instance, $\text{BirthDate} <_{\nu} 11- <_{\nu} 39$ holds iff the BirthDate value is in $[11-01-39, 11-30-39]$. We show some potential inference attacks within the following *query-answer* form:

- $Q1 : \pi_{\{\text{Disease}\}} \sigma_{\{\text{BirthDate} <_{\nu} 11- * - 39 \wedge \text{Zipcode} = 13068\}}$
- $A1 : < \text{Flu} >$ (*Inferred exactly*)
- $Q2 : \pi_{\{\text{Sex}\}} \sigma_{\{\text{BirthDate} <_{\nu} 08- * - 57 \wedge \text{Disease} = \text{B.Cancer}\}}$
- $A2 : < F >$ (*Inferred with about 100%*)

Table 1. A table of health data

	Name	BirthDate	Sex	Zipcode	Disease
1	Lucy	11-12-39	F	13068	Flu
2	Lily	11-02-39	F	13068	Flu
3	Alice	08-24-57	F	14092	B.Cancer
4	Bob	08-02-57	M	13053	Stoke
5	Frank	08-02-42	M	13053	Stoke
6	Jack	11-22-42	M	13053	No
7	Michael	07-25-42	M	13053	AIDS

Table 2. A 2-anonymized table

	BirthDate	Sex	Zipcode	Disease
1	11-*39	F	13068	Flu
2	11-*39	F	13068	Flu
3	08-*57	F	1****	B.Cancer
4	08-*57	M	1****	Stoke
5	**-42	M	13053	Stoke
6	**-42	M	13053	No
7	**-42	M	13053	AIDS

As [12] noted, classical k-anonymity models do not guarantee the sufficient diversity on sensitive attributes, which results in the success of query $Q1$. Another, since it is a common sense that male has much less probability to get B.Cancer (Breast Cancer) than female, query $Q2$ can succeed too. Further, if intruders master some useful information such as the value domains and some anonymization rules on BirthDate, Sex, or Zipcode in Table 1, more attacks can succeed on Table 2.

In this paper, we try to analyze the potential privacy inference attacks on k-anonymized microdata, most of which have not been considered or handled well by current k-anonymity models. We first define the privacy inference logic as a theoretical base, which can be effectively used for capturing various inference attacks and preventing them before microdata publication. It is also our premier contribution in this paper. Beside, we propose a solution to effectively decrease such inference risks which are illustrated by some experiments.

The remainder of the paper is arranged as follows: Section 2 defines the privacy inference logic. Section 3 discusses the k-anonymity model with our proposed

logic. Section 4 analyzes several kinds of knowledge-based inference attacks. Section 5 offers the experiments study to explicitly illustrate these inference attacks and proves the effectivity of our solution. Section 6 discusses some related work on microdata publication and k-anonymity technologies. Finally, Section 7 gives a short conclusion and reveals our future work.

2 Privacy Inference Logic

The basic objects of privacy inference logic are attribute values in a large population. The objective of this logic is to find and formalize the association relations potentially existed between two values or even attributes, including their association strengths, features, and influences on others, etc. So, *Privacy inference logic* which we try to define it in the following paragraphs is an information inference theory evolved from set and probability theories.

Suppose attribute set $A = \{A_1, A_2, \dots, A_m\}$ ($|A| = m \geq 1$) of a microdata, Each attribute A_i ($i = 1, \dots, m$) has a *value domain* D_{A_i} consisting of all possible values that can be appeared on A_i . Any value, value array, or value vector can be seen as an element in the large population Ω . So $\bigcup_{i=1}^m D_{A_i} \subseteq \Omega$. An *instance* of attribute set A , denoted as $a = \{a_1, a_2, \dots, a_m\}$, is to replace each attribute A_i with value a_i in D_{A_i} . The value domain on A is $D_A = \{a | a \text{ is an instance of } A\}$.

Value \tilde{v} is called as an *anonymization form* of value v if it contains less specific information than v ($v, \tilde{v} \in \Omega$) on an attribute. Consequently, v is also called the *original form* of \tilde{v} , if nonexist any value in Ω taking v as an anonymization form. We say there is an *anonymization relation* between v and \tilde{v} , denoted as $v <_\nu \tilde{v}$. A value may have several anonymization forms containing different degrees of information.

The anonymization relation is *transitive*. If $v <_\nu \tilde{v}$, $\tilde{v} <_\nu \tilde{\tilde{v}}$, then $v <_\nu^+ \tilde{\tilde{v}}$ holds, where $<_\nu^+$ indicates that there exists a “middle” anonymized value list $\{\tilde{v}^1 <_\nu \dots <_\nu \tilde{v}^n\}$ ($n \geq i$) satisfying $v <_\nu \tilde{v}^1 <_\nu \dots <_\nu \tilde{v}^n <_\nu \tilde{\tilde{v}}$.

For convenience, we define $v <_\nu^* \tilde{v} = v <_\nu \tilde{v} \cup v <_\nu^+ \tilde{v}$, and say v *satisfies* \tilde{v} or \tilde{v} is *satisfied by* v , denoted as v st. \tilde{v} , iff $v <_\nu^* \tilde{v}$ holds.

We define the *super value domain* on A_i ($i = 1, \dots, m$) as $D_{A_i}^* = D_{A_i} \cup \{\tilde{v}_i | \exists v_i \in D_{A_i}, v_i <_\nu^* \tilde{v}_i\}$. A *super instance* of A is defined as $sa = \{a_1, a_2, \dots, a_m\}$, where $a_i \in D_{A_i}^*$. And the super value domain on A is $D_A^* = \{sa | sa \text{ is a super instance of } A\}$. sa is called an *anonymization instance* (or *form*) of A if $\exists a_i \in sa$ ($1 \leq i \leq m$), $\exists a_i^0 \in D_{A_i}, a_i^0 <_\nu^+ a_i$.

As an extension, an *anonymization form* of value set $V = \{v_1, v_2, \dots, v_l\}$ ($l \geq 1$) is a set consisting of at least one anonymization form of a value in V , denoted as $\tilde{V} = \{\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_l\}$, satisfying $\exists i, v_i <_\nu^+ \tilde{v}_i (1 \leq i \leq l) \wedge \forall j <_\nu^* \tilde{v}_j (j = 1, \dots, l)$.

Suppose $D_A^* = \{sa_1, sa_2, \dots, sa_n\}$, ($n \geq 1$), $a_i \in sa_k$, $a_j \in sa_k$ ($1 \leq i, j \leq m, 1 \leq k \leq n$), we say there is an inference relation from a_i to a_j , if a_j can be ascertained w.r.t. a_i in some sense. The certainty degree is represented by what is called inference probability in the range between 0 and 1. Specifically, the

inference probability from \tilde{v} to v (on $v <_{\exists} \tilde{v}$) in scope Ω denoted as $P(\tilde{v} \hookrightarrow v|_{\Omega})$, indicates the likelihood of correctly guessing v when knowing \tilde{v} in Ω as: $P(\tilde{v} \hookrightarrow v|_{\Omega}) = \frac{1}{V_{\Omega}}$, where $V_{\Omega} = \{v' | v' \in \Omega, v' <_{\nu}^* \tilde{v}\}$. Similarly, we define it in scope D_v ($v \in D_v \subseteq \Omega$) as: $P(\tilde{v} \hookrightarrow v|_{D_v}) = \frac{1}{V_{D_v}}$, where $D_v = \{v' | v' \in D_v, v' <_{\nu}^* \tilde{v}\}$. Obviously, the following formula holds.

$$P(\tilde{v} \hookrightarrow v|_{\Omega}) \leq P(\tilde{v} \hookrightarrow v|_{D_v}) \quad (1)$$

The above scopes can be replaced by others, such as the *super value domain* D_v^* , etc. Beside, we can also easily extend the two notions on value vectors.

Suppose a partition on A , A' and A'' satisfy $A' \cup A'' = A$, $A' \cap A'' = \emptyset$. We say there is an *attribute matching* relation between A' and A'' in scope D_A (similar to that in scope D_A^*), denoted as $A' - A''$, iff $\forall a' \in D_{A'} (\text{or } a'' \in D_{A''}), \{a', a''\} \in D_A$. a' and a'' is a *value matching*.

On an inference relation from \tilde{v} to v , we define the *inference attack* on it as the behavior to infer v w.r.t. \tilde{v} with a certain success probability. To precisely measure it, we define a *threshold* δ ($0 \leq \delta \leq 1$) as the lower bound of the inference probability for a *successful* inference attack.

Since an attribute or its value is often in some associations with others. On an inference relation, an intruder may infer the objects (identifier or sensitive attribute values) by virtue of these associations known. In this logic, we define *knowledge* as a generic concept to cover all useful information (i.e. inference associations existed except the inference relation) that may be used by intruders to successfully infer the desired objects. If an inference attack cannot be successful in normal situation but can be *successful* under some prior information(i.e. when intruders master some knowledge and use them during the inference attack behavior), it is called a *(successful) knowledge-based inference attack*.

For instance, suppose there are three inference relations from v_i to v_k , from v_i to v_j , from v_j to v_k , respectively. If an intruder wants to infer v_k on the first inference relation with enough high precision, the other two inference relations may be seen as useful knowledge in the intruder's view, since they may increase the inference probability as to make the intruder know that $P(v_i \hookrightarrow v_k) \geq P(v_i \hookrightarrow v_j) \times P(v_j \hookrightarrow v_k)$. Such a transitive inference probability calculation can be extended to a set of "middle" value lists beginning from v_i and ending with v_k with a sequence of inference relations available for calculating $P(v_i \hookrightarrow v_k)$.

If the ratio of the inference probability on an inference attack under knowledge c to the probability under no knowledge is larger than threshold ω ($\omega \geq 1$), we call the inference a *relative (successful) inference attack under c*. For instance, if $\frac{P(\tilde{v} \hookrightarrow v|_{D_v})}{P(\tilde{v} \hookrightarrow v|_{\Omega})} \geq \omega$, it is a relative inference attack to infer the exact value v w.r.t. \tilde{v} under knowledge D_v .

One special and complicated knowledge-based inference attack is what is called *conditional inference attack*. Suppose attribute set $A = A' \cup A''$, $a'_i, a'_j \in D_{A'}, a''_i, a''_j \in D_{A''}$, $\{a'_i, a''_i\}, \{a'_j, a''_j\} \in D_A$. $P(a'_i \hookrightarrow a''_i | a'_j \hookrightarrow a''_j)$ refers to the inference probability from a'_i to a''_i under knowledge of $P(a'_j \hookrightarrow a''_j|_{D_A})$. We say it is a *successful* conditional inference attack, if one of the two conditions holds:

$$(1) P(a'_i \hookrightarrow a''_i | a'_j \hookrightarrow a''_j) \geq \text{Max}(P(a'_i \hookrightarrow a''_i|_{D_A}), \delta); \quad (2) \frac{P(a'_i \hookrightarrow a''_i | a'_j \hookrightarrow a''_j)}{P(a'_i \hookrightarrow a''_i|_{D_A})} \geq \omega.$$

We take an example to illustrate such notion. Suppose two sets $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3\}$. If we just have the knowledge that there is an *one-to-one* mapping relation between X and Y , we can infer $P(x_{1|2|3} \hookrightarrow y_{1|2|3}) = \frac{1}{3}$ ($1|2|3$ indicates any one of three). But if we have another knowledge that $P(x_2 \hookrightarrow y_2) = \frac{2}{3}$, we can infer $P(x_{1|3} \hookrightarrow y_2 | x_2 \hookrightarrow y_2) = P(x_2 \hookrightarrow y_{1|3} | x_2 \hookrightarrow y_2) = \frac{1}{6}$, $P(x_{1|3} \hookrightarrow y_{1|3} | x_2 \hookrightarrow y_2) = \frac{5}{12}$.

3 Formulation of K-Anonymity Model

In this section, we discuss k-anonymity with the privacy inference logic. Suppose individual data which are recorded in a microdata set include *unique identity* attributes (UI), *quasi-identifier* attributes (QI), and *sensitive* attributes (SI) [17]. For example, in Table 1, Name is an UI attribute, {BirthDate, Sex, Zipcode} are the QI attributes, and Disease is a SI attribute. As discussed in Section 1, k-anonymity model is to derive an anonymized microdata from the original one, in which each record is indistinguishable from at least $k-1$ other records with respect to the QI attributes. This requirement is satisfied by Table 2 when $k = 2$, which may be published in general k-anonymity applications, even if we have discovered some successful inference attacks on it as above illustrated.

For convenience, we use \mathcal{T} and \mathcal{T}' represent the original and anonymized microdata (with n records and m attributes excluding UI attributes), respectively. $|QI| = q$, $|SI| = s$, $q + s = m$. And $QCluster$ and $SCluster$ denote the QI tuple cluster mapping to a SI tuple and the SI tuple cluster mapping to a QI tuple in \mathcal{T}' . They are denoted as $QCluster_i$ and $SCluster_i$ on the i^{th} record respectively. For example, in Table 2, $SCluster_1 = \{\text{Flu, Flu}\}$, $QCluster_1 = \{<11-*39, F, 13068>, <11-*39, F, 13068>\}$. Through the k-anonymization process on \mathcal{T} , we can get an k-anonymized table \mathcal{T}' , in which any tuple on QI attributes is the anonymization form \widetilde{QI} of the original form in \mathcal{T} , i.e. $\forall i, 1 \leq i \leq n; q_i \sim \widetilde{q}_i$, where q_i and \widetilde{q}_i are the i^{th} tuple on QI in \mathcal{T} and \mathcal{T}' , respectively, or more essentially, q_i is the original form of \widetilde{q}_i . In the privacy inference logic, \mathcal{T} is a set of original instances of attribute set $QI \cup SI$ with $D_{QI \cup SI}$, while \mathcal{T}' is a set of the corresponding super instances with $D_{QI \cup SI}^*$.

The primary goal for k-anonymization is to prevent individual re-identification, and ultimately, to prevent the successful inference attacks on UI-SI matching in \mathcal{T} when \mathcal{T}' is published. Within the privacy inference logic framework, it can be described as follows:

$$\underbrace{\mathcal{U}I}_{In \quad \mathcal{T}} \xleftarrow{\mathcal{P}_{\mathcal{T}}(QI \hookrightarrow \mathcal{U}I)} \underbrace{QI}_{Anonymization} \xleftarrow{\mathcal{P}_{\mathcal{T}}(\widetilde{QI} \hookrightarrow QI)} \underbrace{\widetilde{QI}}_{In \quad \mathcal{T}'} \xrightarrow{\mathcal{P}_{\mathcal{T}'}(\widetilde{QI} \hookrightarrow SI)} \underbrace{SI}_{(2)}$$

$$\mathcal{P}_{\mathcal{T} \& \mathcal{T}'}(\mathcal{U}I \leftrightarrow SI) = \mathcal{P}_{\mathcal{T}}(QI \hookrightarrow \mathcal{U}I) \times \mathcal{P}_{\mathcal{T}}(\widetilde{QI} \hookrightarrow QI) \times \mathcal{P}_{\mathcal{T}'}(\widetilde{QI} \hookrightarrow SI) \quad (3)$$

$\mathcal{P}_{\Omega}(QI \hookrightarrow \mathcal{U}I|_{\Omega})$ is calculated within Ω . For instances, in Cambridge, Massachusetts in 1997, the following hold (Ω may be the whole Cambridge city, $\mathcal{U}I = \{\text{Name, Address}\}$, $QI = \{\text{BirthDate, Sex, Zipcode}\}$): $\mathcal{P}_{\Omega}(\{\text{BirthDate, Sex, Zipcode}\} \hookrightarrow \{\text{Name, Address}\}) \geq 0.97$ [15]. To make it more significant and

operable in practice, we define $\Omega = \mathcal{T}$. Within the i^{th} record of \mathcal{T} , $\mathcal{P}_{\mathcal{T}}(qi_i \hookrightarrow ui_i)$ can be calculated as follows (qi_i and ui_i are the i^{th} tuples of QI and UI in \mathcal{T}).

$$\mathcal{P}_{\mathcal{T}}(qi_i \hookrightarrow ui_i | \mathcal{T}) = \frac{1}{|\mathcal{T}_{qi}^{st. qi_i}|} \quad (4)$$

Where $\mathcal{T}_{qi}^{st. qi_i}$ is the cluster of qi value tuples in \mathcal{T} satisfying (i.e. equal to in \mathcal{T}) qi_i . For instances, in Table 1, $P(\{Sex = F\} \hookrightarrow \{Name = Lucy\}) = \frac{1}{3}$, $P(\{Zipcode = 13053\} \hookrightarrow \{Name = Frank\}) = \frac{1}{4}$, $P(Sex \hookrightarrow Name) = \frac{2}{7}$.

We define: $\mathcal{P}_{\mathcal{T}}(\mathcal{QI} \hookrightarrow \mathcal{UI} | \mathcal{T}) = \frac{1}{n} \cdot \sum_{i=1}^n P_{\mathcal{T}}(qi_i \hookrightarrow ui_i)$. As on Table 2, $\mathcal{P}(\{Sex, Zipcode\} \hookrightarrow Name) = \frac{7}{12}$, $\mathcal{P}(\{BirthDate, Sex, Zipcode\} \hookrightarrow Name) = 1$, etc.

Since the i^{th} QI tuple \widetilde{qi}_i in \mathcal{T}' is the *anonymization form* of qi_i in \mathcal{T} . On $qi_i <_{\nu}^* \widetilde{qi}_i$, the inference probability $P(\widetilde{qi}_i \hookrightarrow qi_i | \mathcal{T})$ can be calculated as:

$$\mathcal{P}(\widetilde{qi}_i \hookrightarrow qi_i) = \frac{1}{|\mathcal{T}_{qi}^{st. \widetilde{qi}_i}|} \quad (5)$$

Where $\mathcal{T}_{qi}^{st. \widetilde{qi}_i}$ is the cluster of qi value tuple in \mathcal{T} satisfying \widetilde{qi}_i in \mathcal{T}' (i.e. $qi <_{\nu}^* \widetilde{qi}_i$). $\mathcal{P}_{\mathcal{T}}(\widetilde{\mathcal{QI}} \hookrightarrow \mathcal{QI})$ is the mean of $P_{\mathcal{T}}(\widetilde{qi}_i \hookrightarrow qi_i)$ on the whole \mathcal{T} scope. It can be noted that the above formula is to calculate the real inference probability when users capture knowledge \mathcal{QI} value domains in \mathcal{T} . For instances, $P(BirthDate = 11-*39 \hookrightarrow BirthDate = 11-12-39) = \frac{1}{2} \neq \frac{1}{30}$, $P(\{Sex = *, Zipcode = 1 * ***\} \hookrightarrow \{Sex = F, Zipcode = 14092\}) = \frac{1}{7}$.

As each \widetilde{qi}_i in \mathcal{T}' ($1 \leq i \leq n$) has at least $k-1$ duplicated tuples. The inference probability from \widetilde{qi}_i to si_i , denoted as $P(\widetilde{qi}_i \hookrightarrow si_i | \mathcal{T}')$, can be calculated w.r.t. the $SCluster_i$ size. $P(\widetilde{QI} \hookrightarrow SI | \mathcal{T}')$ is its mean value.

$$P(\widetilde{qi}_i \hookrightarrow si_i | \mathcal{T}') = \frac{|si| si \in SCluster_i, si <_{\nu}^* si_i|}{|SCluster_i|} = \frac{\mathcal{T}'_{\widetilde{qi}_i}^{st. \widetilde{qi}_i} \cap \mathcal{T}'_{si}^{st. si_i}}{\mathcal{T}'_{qi}^{st. \widetilde{qi}_i}} \quad (6)$$

As in Table 2, $\mathcal{P}(\{BirthDate = 08-*57, Sex = *, Zipcode = 1 * ***\} \hookrightarrow Disease = Stoke) = \frac{1}{2}$, $\mathcal{P}(\{BirthDate = **-42, Sex = M, Zipcode = 13053\} \hookrightarrow Disease = Stoke) = \frac{1}{3}$.

The purpose for k-anonymity is to make $\mathcal{P}(ui_i \hookrightarrow si_i | \mathcal{T} \& \mathcal{T}')$ less than a pre-defined threshold δ on the i^{th} ($1 \leq i \leq n$) record of \mathcal{T}' . Based on the above analysis, the way to reach it is to keep each “middle” inference probability between ui_i and si_i under a threshold, including $P(\widetilde{qi}_i \hookrightarrow qi_i)$, and $\mathcal{P}(\widetilde{qi}_i \hookrightarrow si_i | \mathcal{T}')$.

4 Privacy Inference Attacks

As the above analyzed, general k-anonymity models can make inference risks under a controllable scope. But when considering some knowledge, inference risks may be increased greatly from the perspective of intruders.

4.1 Simple Privacy Inference Attacks

Simple privacy inference attack is to recognize individual identity on anonymized QCluster tuples, or to infer some QCluster-SCluster associations with enough precision when no knowledge considered. In other words, it is to discover some information that should be concealed by the anonymization w.r.t just \mathcal{T}' . Although it is caused by indeliberately ignoring some special situations on the anonymized microdata when anonymizing the original one (such as a SCluster contains just one element), this type attacks can be discovered and prevented when adjusting the anonymization algorithm to consider these special requirements.

For instance, in Table 2, it is easily discovered that the exact Disease value on the 1st record mapping to $\langle 11\text{-}\ast\text{-}39, \text{F}, 13068 \rangle$ is Flu because of the mapped SCluster just contains this single element, which is likely to increase $P(qi_1 \hookrightarrow si_1)$ to 1. The main reason is the diversity degree of SCluster in \mathcal{T}' which is not enough to prevent inferring an exact value in it. To avoid this attack, we can define a threshold ℓ , and take $Diversity(\text{SCluster}) \geq \ell$ as an anonymization rule effectively embedded into the anonymization process, where $Diversity(\text{SCluster})$ refers to the diversity of a SCluster w.r.t. a measurement [T2].

4.2 Knowledge-Based Privacy Inference Attacks

In practice, an anonymized microdata is published into some public databases for statistical, general analysis. One big problem for microdata publishing while preventing personal privacy disclosure is: it is hard to ascertain or suppose what knowledge the intruder may hold. There are three main reasons: (1)Relative knowledge available for attacks cannot be tractable; (2)Intruder group in reality is too large and diverse to be computable; (3) To prevent inference attacks is not always consistent with the anonymization rules.

We conclude these features of knowledge as its *global intractability*. But when focusing on an anonymized microdata, we can capture the most of knowledge (*main knowledge*) used for knowledge-based inference attacks. Such a feature of knowledge is called *local computability*.

We divide knowledge-based privacy inference attacks into three classes: (1) Value inference attack under value domains known, which is to infer the exact value on an attribute when knowing QI value domains; (2) Conditional inference attack under value associations known, which is to infer the more precise value associations when knowing some value associations; (3) QI tuple inference attack under anonymization rules and relations known, which is to infer the original or more specific QI tuples when knowing some anonymization rules and relations.

For the first class of knowledge-based inference attacks, it is obvious that the probability of inferring value v in value domain D_v is much larger than that in the population Ω as there are many instances above proving it. As the Formula (1), the intruder can get a higher probability from \tilde{v} to v in D_v than in Ω . The essential reason for this attack is $D_v \subseteq \Omega$ and D_v is tractable in most situations. For example, suppose the intruder has known: $D_{BirthDate} = \{11\text{-}12\text{-}39, 11\text{-}02\text{-}39, 08\text{-}24\text{-}57, 08\text{-}02\text{-}57, 08\text{-}02\text{-}42, 11\text{-}22\text{-}42, 07\text{-}25\text{-}42\}$ in \mathcal{T} . The following inference probabilities (on BirthDate column in \mathcal{T}') hold to the intruder:

- $P(11\text{-}\ast\text{-}39 \hookrightarrow 11\text{-}12\text{-}39|11\text{-}02\text{-}39) = \frac{1}{2} \neq \frac{1}{30}$
- $P(\ast\text{-}\ast\text{-}42 \hookrightarrow 08\text{-}02\text{-}42|11\text{-}02\text{-}42|07\text{-}25\text{-}42) = \frac{1}{3} \neq \frac{1}{12 \times 30}$

In short, to know the real value domain on an attribute is helpful to narrow the scope used to infer the exact value. It should not be ignored for microdata publication since in some situations the intruder can hold these prior information.

For the second class of knowledge-based inference attacks, to capture all value associations among two or several attributes is difficult. Same to destroy a value association contained in QCluster-SCluster in \mathcal{T}' , as it needs the anonymization process to find all value associations and to avoid the successful inference attacks in the QCluster-SCluster. For example, it is hard to discover the value association Disease=B.Cancer \rightarrow Sex=M [0%] in $\{08\text{-}\ast\text{-}57, \ast, 1\ast\ast\ast\} - \{\text{B.Cancer, Stoke}\}$ when anonymizing \mathcal{T} . Even if it is discovered before anonymization, it is indispensable for the anonymization process to guarantee the inference probabilities in the matching smaller than the threshold.

For the third class of knowledge-based inference attacks, since anonymization rules and relations for general k-anonymity models are always static and simple, they are easily captured by intruders. For example, it is easy to know the k value, the generalization hierarchy or some suppression techniques on some attributes used for the anonymization process through observing and analyzing the published microdata.

If intruders know several kinds of integrated knowledge, for inference, including some value domains on some attributes, some value associations, and anonymization rules used for anonymization, the inference attack may be more complicated to be prevented.

For instance, if $D_{Zipcode}$ and its anonymization rules are known, it can be inferred the original values from $1\ast\ast\ast$ are in $\{14092, 13053\}$, but not 13068, since there are just two 13068 in the value domain, and they have been appeared in \mathcal{T}' , i.e. $P(1\ast\ast\ast \hookrightarrow 14092|13053) = \frac{1}{2}$. So if the intruder can know Disease=B.Cancer \rightarrow Sex=M [0%] and $P(\text{Zipcode}=14092 \hookrightarrow \text{Name}=Alice) = 1$, $P(\text{Disease}=B.\text{Cancer} \hookrightarrow \text{Name}=Alice) = \frac{1}{2}$ can hold. With similar knowledge, the intruder can infer: $P(\text{Name}=Lucy|\text{Lily} \leftrightarrow \text{Disease}=Flu) = \frac{1}{2}$.

5 Experiment Study

Same to [9][12], the initial microdata set in the experiment is the Adult database supported by UCI [19]. We adopt the training dataset containing 4 QI attributes (*Age, Sex, Education, and Country*) and 2 SI attributes (*Salary and Occupation*).

As the above discussed, the main knowledge available for privacy inference attacks on k-anonymized microdata include value domains, value associations, and anonymization rules and relations. We suppose the population is the Adult Database (\mathcal{T}). Beside, we take an Apriori¹ algorithm to create a set of value inference relations in \mathcal{T} with high inference probability ($\geq 90\%$). Under the

¹ Apriori is a classic data mining algorithm to find association rules in a database with enough confidence and supportance [5].

knowledge of QI value domains, the inference probability on $\tilde{qi}_i \hookrightarrow qi_i$ can be calculated as: $P(\tilde{qi}_i \hookrightarrow qi_i) = \text{Max}(P_0(\tilde{qi}_i \hookrightarrow qi_i), P_1(qi_i \hookrightarrow \tilde{qi}_i))$, where $P_0(\tilde{qi}_i \hookrightarrow qi_i)$ is the inference probability under empty knowledge on $qi_i \hookrightarrow qi_i$, $P_1(qi_i \hookrightarrow \tilde{qi}_i) = \frac{1}{|\mathcal{T}'_{qi_i}^{st.}|}$.

Table 3. Some information derived when $k = 2$

(k, β_{us})	(2, 0.1)	(2, 0.2)	(2, 0.3)	(2, 0.4)
$\overline{p^0}_{u-s_0}$	0.185(238)	0.284(74)	0.406(16)	0.5(7)
$\overline{p^0}_{u-s_1}$	0.186(339)	0.282(114)	0.433(20)	0.5(12)
$\overline{p^0}_{u-s_2}$	0.234(1292)	0.285(863)	0.476(135)	0.5(112)
$\overline{p^0}_{u-s_3}$	0.241(1461)	0.298(940)	0.489(194)	0.51(168)
$\overline{p^1}_{u-s_0}$	0(0)	0(0)	0(0)	0(0)
$\overline{p^1}_{u-s_1}$	0(0)	0(0)	0(0)	0(0)
$\overline{p^1}_{u-s_2}$	0.207(82)	0.261(47)	0.5(2)	0.5(2)
$\overline{p^1}_{u-s_3}$	0.206(83)	0.261(47)	0.5(2)	0.5(2)

In Table 3, $\overline{p^0}_{u-s_0}$, $\overline{p^0}_{u-s_1}$, $\overline{p^0}_{u-s_2}$ and $\overline{p^0}_{u-s_3}$ stand for $P(UI \hookrightarrow SI)$ under no knowledge, knowledge of value domains, value domains and anonymization rules and relations, and value domains and anonymization rules and relations and value associations, respectively, when $k = 2$. Similarly, $\overline{p^1}_{u-s_0}$, $\overline{p^1}_{u-s_1}$, $\overline{p^1}_{u-s_2}$ and $\overline{p^1}_{u-s_3}$ are the counterpart values derived from the k-anonymization process with our solution: *a new anonymization cost metric considering inference risk (probability) under kinds of knowledge – greatly increasing the metric value when the risk is enough high, while keeping it uninfluenced when the risk is tolerable.*

It is obvious that the amount of successful inference attacks under main knowledge is decreased. With this metric, the inference attacks under empty knowledge are prevented fully, as there are 238 records satisfying $P(ui \leftrightarrow si) \geq \delta = 0.1$ but 0 record by the metric.

Fig.1 describes the inference probability under SI value domains on the first 500 records in a 2-anonymized microdata, which illustrates that the SI value domains

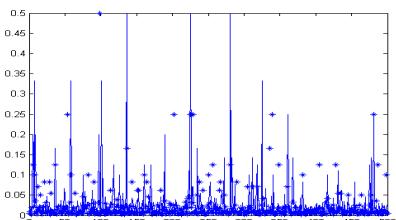


Fig. 1. (Left): $P(\tilde{QI} \hookrightarrow QI)$ and $P(QI \hookrightarrow \tilde{QI})$ of first 500 2-anonymized records

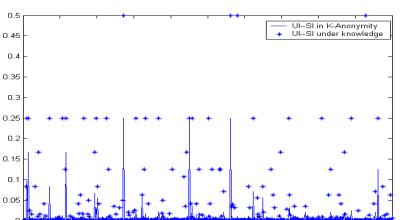


Fig. 2. (Right): $P(UI \hookrightarrow SI)$ in general k-anonymity model and under main knowledge

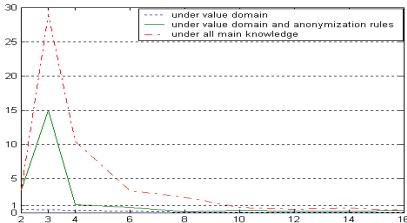


Fig. 3. (Left): The relative increasing ratio of $P(UI \leftrightarrow SI)$ under knowledge with the anonymization cost metric on generalization height

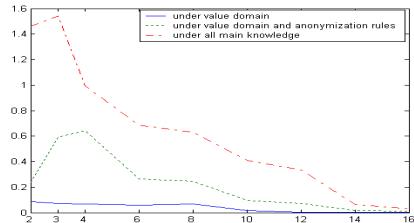


Fig. 4. (Right): The relative increasing ratio of $P(UI \leftrightarrow SI)$ under knowledge with the anonymization cost metric considering inference risk

knowledge is much helpful for increasing inference probabilities on most records. Fig.2 illustrates that the UI-SI matching probability $P(UI \leftrightarrow SI)$ is increased under all main knowledge on many records. When k increases, the average relative increasing ratio of inference probabilities under different knowledge (i.e. value domain, anonymization rules and relations, etc.) to that under empty knowledge, $\frac{P(UI \leftrightarrow SI|_c) - P(UI \leftrightarrow SI)}{P(UI \leftrightarrow SI)}$ (c is a kind of knowledge), is illustrated in Fig.3. And Fig.4 illustrates the relative increasing ratio of inference probabilities under knowledge with the new metric, comparing to that in Fig.3, which is obviously decreased.

6 Related Work

K-anonymity is concerned for identifier disclosure with the modification of microdata containing confidential information about individuals. Information loss is the quantity of information that existed in the initial microdata but does not occur in anonymized microdata because of disclosure control methods [4][7]. There are many information loss metrics proposed in literature, such as the five kinds of information loss metrics summarized in [8]. In [4][8], the record-level disclosure risk is analyzed through computing the probability of matching an exact individual to any record in the initial microdata under many assumptions.

K-anonymity focuses on how to prevent individual re-identification on anonymized microdata and at the same time keep the integrity of the data in the modified microdata by generation/supresion techniques. It is proven that the k-anonymity problem is *NP-hard* even when the attribute values are ternary [1][3]. An algorithm with $O(k)$ -approximation is proposed for this problem [1]. Violation of k-anonymity occurs when a particular attribute value of an entity can be determined to be among less than k possibilities by using the views together with the schema information of the private table [2]. It is proved that whether a set of views occur privacy disclosure for k-anonymity is also a computationally hard problem [2].

[2] proposed ℓ -diversity mechanism to improve the anonymity effect of modified microdata. It is to set and increase the *diversity* value (i.e. the information entropy value evaluation of a cluster of sensitive values) on each sensitive value cluster, ensuring it satisfy a threshold. Comparing to ℓ -diversity, our approach for privacy inference control is more extended and strict. As the inference logic gives a probabilistic way to evaluate not only the re-identification disclosure risks, but also the predictive disclosure risks based on the information containing in original and published microdata sets, and especially, which can be more predicted before publication.

Privacy violated on the relations among data is deeply discussed in [7], in which the disclosure risk is determined directly by the re-identification probability between a record and its respondent. The idea is extended in this paper. We evaluate the re-identification value (when the identity and the sensitive values coexist) as well as the inference probability between re-identifying and sensitive values. Especially, the approach in the paper allows individuals to set their privacy preference to some extend, i.e. the threshold of privacy protection (or inference), which can be implemented as the experiments illustrated above.

Anonymization cost metric mechanisms in most k-anonymity models can be classified into four types w.r.t. the detail computing base [10]: (a) on generalization hierarchy (and generalization height), as Sweeney's *Prec* calculation [17] and Aggarwal's *Cost* computation [1]; (b) on the amount of suppressed cells (such as distance among tuples), as Aggarwal's *Hamming distance* computation [1] and Meyerson's *Diameter* calculation [13]; (c) on partition, as Lyengar's *CM* formula [11], Bayardo's *DM*, *CM* formulas [2], and Lefevre's *C_{AVG}* formula [8]; and (d) on *entropy*, as Machanavajjhala's *Entropy* calculation [12] and Fung's *Entropy* formula [3]. In k-anonymity models, anonymization loss metrics should measure the real information loss under some knowledge, which should be directly relative with the strength of preventing inference attacks.

7 Conclusion

In conclusion, we analyzed some privacy inference attacks that may be potentially existed in anonymized dataset based on the inference theory on attributes and its values. To solve these kinds of inference attacks should depend on a more reasonable anonymization cost metric for k-anonymity, which should reflect the true information loss on the anonymized microdata under some complicated assumptions. Our future work on this topic is to define a well-designed k-anonymity model that can prevent the above discovered information attacks with more precise quantitative information probability calculations.

Acknowledgements

This work was partial supported by the National Natural Science Foundation of China under Grant No.60673140.

References

1. Gagan Aggarwal and Tomas Feder and etc. Approximation Algorithms for K-Anonymity, Journal of Privacy Technology, Paper number: 20051120001, Nov. 2005.
2. R. J.Bayardo and R. Agrawal: Data Privacy through Optimal K-Anonymization, Proc. of ICDE 2005, 217-228, Tokyo, Japan.
3. Benjamin C.M. Fung, Ke Wang and Philip S.Yu: Top-Down Specialization for Information and Privacy Protection, Proc. of ICDE 2005,205-216, Tokyo, Japan.
4. William E.Winkler: Re-Identification Methods for Masked Microdata, Privacy in Statistical Databases 2004, 216-230,Barcelona, Spain.
5. Jiawei Han and Micheline Kamber: Data Mining Concepts and Techniques, Magan Kanfmann Publishers, August 2000.
6. Daniel Kifer and Johannes Gehrke: Injecting Utility into Anonymized Datasets, Proc. of SIGMOD 2006, 217-229, Chicago, USA.
7. Diane Lambert: Measures of Disclosure Risk and Harm, <http://cm.bell-labs.com/cm/ms/departments/sia/doc/93.17.ps>.
8. Kristen LeFevre, David J.DeWitt and Raghu Ramakrishnan: Multidimensional K-Anonymity, Technical Report, <http://www.cs.wisc.edu/techreports/2005/>
9. Kristen Lefevre, David J.DeWitt and Raghu Ramakrishnan: Incognito: Efficient Full-Domain K-Anonymity, Proc. of SIGMOD 2005,49-60,Baltimore, USA.
10. Zude Li, Guoqiang Zhan and Xiaojun Ye: Towards a More Reasonable Generalization Cost Metric For K-Anonymization, Proc. of BNCOD 2006, 258-261, Belfast, UK.
11. Vijay S. Lyengar: Transforming Data to Satisfy Privacy Constraints, Proc. of SIGKDD 2002, 279-288,Edmonton, Canada.
12. Ashwin Machanavajjhala, Johannes Gehrke and Daniel Kifer: ℓ -Diversity: Privacy Beyond K-Anonymity, Proc. of ICDE 2006,Atlanta, USA.
13. Adam Meyerson and Ryan Williams: On the Complexity of Optimal K-Anonymity, Proc. of PODS 2004, 223-228, Paris,France.
14. Pierangela Samarati and Latanya Sweeney: Protecting Privacy when Disclosing Information: K-Anonymity and Its Enforcement Through Generalization and Suppression, Technical Report, SRI Computer Science Lab., 1998, <http://privacy.cs.cmu.edu/people/sweeney/publications.html>.
15. Latanya Sweeney: Guaranteeing Anonymity when Sharing Medical Data, the Datafly System, Journal of the American Medical Informatics Association, 1997, http://adams.mgh.harvard.edu/PDF_Repository/D004462.PDF.
16. Latanya Sweeney: Achieving K-Anonymity Privacy Protection Using Generalization and Suppression, Intl. Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10(5),571-588, 2002.
17. Latanya Sweeney: K-Anonymity: A Model For Protecting Privacy, Intl. Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10(5),557-570, 2002.
18. Traian Marius Truta, Farshad Fotouhi and Daniel Barth-Jones: Disclosure Risk Measures for Microdata, Proc. of the 15th Intl. Conf. on Scientific and Statistical Database Management, 2003,15-22,Cambridge, USA.
19. UCI. U.c. irvine machine learning repository. <http://www.ics.uci.edu/mlearn>.
20. Willemborg and L. Waal, Elements of Statistical Disclosure Control. Springer Verlag, 2001.
21. Chao Yao, X. Sean Wang and Sushil Jajodia: Checking for K-Anonymity Violation by Views, Proc. of VLDB 2005, 910-921, Trondheim,Norway.

Schema Mapping in P2P Networks Based on Classification and Probing

Guoliang Li^{1,*}, Beng Chin Ooi², Bei Yu², and Lizhu Zhou¹

¹ Department of Computer Science and Technology

Tsinghua University, Beijing 100084, China

{liguoliang,dcszlz}@tsinghua.edu.cn

² School of Computing,

National University of Singapore, Singapore

{ooibc,yubei}@comp.nus.edu.sg

Abstract. In this paper, we address the problems of adaptive schema mappings between different peers in peer-to-peer networks and searching for interesting data residing at different peers based on such mappings. We begin by classifying the shared schema of each peer into a taxonomy of relation categories and attribute categories. We then propose our adaptive schema mapping by selectively probing the shared schema with query probes, which are generated by the classification rules. To improve the accuracy of schema mapping, we introduce the notion of confusion matrix and prior-knowledge. Finally, we present the query reformulation strategy for retrieving and integrating data from all relevant peers. We have implemented our proposed schema mapping and query processing methods in real settings with real datasets. The experimental results show that our method can be adopted effectively in practice.

1 Introduction

Sharing data among multiple sources is crucial in a wide range of applications, including enterprise data management, large-scale scientific projects, government agencies and the World-Wide Web in general. Data integration approaches offer an architecture for data sharing in which data is queried through a mediated schema, but physically stored at the source locations based on their own schemas. Recent data integration systems have been successful at enabling data sharing, but on a relatively small scale, due to the expensive cost of constructing the mediated schema.

Recently, peer data management systems (PDMS) have been proposed as an architecture for decentralized data sharing [1,2,9,19,20,23]. A PDMS consists of a set of (physical) peers, and each peer has an associated schema, denoted as *peer schema*, that represents its domain of interest. Some peers store actual data with mappings between their physical schemas to their relevant peer schemas. However, a peer may not have complete data instances for its peer schema, since individual peers typically do not contain complete information about a domain.

* The work was done when the author was on internship attachment at the National University of Singapore.

This calls for schema mappings in order to tap on relevant peers for more complete answers. Mapping all data sources to a single global schema (or mediator) in a PDMS is not feasible due to the decentralization and scalability requirements of P2P systems. Therefore, in a PDMS, mappings between disparate schemas are built directly and stored locally, such that when a query is posed at a peer, the answers are obtained by integrating retrieved results of reformulated queries from relevant peers, which are generated by exploring the mappings.

Schema mapping of most existing proposals for PDMS such as Hyperion [2][5], Piazza [9][23], and PeerDB [9][20] all require human intervention, which is inefficient and ineffective for large networks and dynamic sources. Therefore, an adaptive way for generating schema mapping is highly desirable. In this paper, we propose such a schema mapping method based on classification. We classify the shared schemas (relational tables and attributes) of individual peers into a taxonomy of *relation categories* and associated *attribute categories*, which essentially represent various conceptual domains. For all peers that have relations belonging to the same category, schema mappings are generated for them. When a new peer joins, classification of its shared schema is performed by probing its relations with query probes generated from classification rules, and consequently, it will be assigned to one or more relation categories to which the probing results have best matches. Subsequently, its schema is mapped to peers in the same categories.

The advantage of our classification-based schema mapping is that its simplicity and modeling uniformity allow integrating the contents of several sources without having to tackle complex structural differences. Another advantage is that query evaluation in classification-based sources can be done efficiently.

Our system is based on a super-peer P2P network in which super peers themselves are organized in a structured overlay, such as BATON [12], and normal peers within the cluster managed by a super peer are unstructured. The categories are distributed among super peers, through which normal peers build schema mappings. Our categories structure is distinct from a global schema (or mediator), since it is distributed among all the super peers, and it is used for peers to generate schema mappings, not for users to pose queries.

In this paper, we make the following contributions. First, we propose a method for schema mapping based on classification and probing in PDMS. Second, we adopt the notion of confusion matrix [16] and apply prior-knowledge to improve the accuracy of schema mapping whenever there are overlapping instances among the shared schemas. Third, we present query formulation strategies for reformulating local queries among relevant peers to achieve efficient query answering.

The paper is organized as follows. We discuss the related work in Section 2. Section 3 presents how to create the schema mapping, and Section 4 describes the query reformulation and evaluation strategies. In Section 5, we provide extensive experimental evaluations of our method and we conclude the paper in Section 6.

2 Related Work

There is no doubt a long stream of research on schema mapping, and we shall briefly review recent and relevant proposals. Kang et al. [14] investigated schema

matching techniques that worked in the presence of opaque column names and data values. Yu et al. [27] proposed a method about constraint-based XML data integration. Dhamankar et al. [4] described the iMAP system which semi-automatically discovered both 1-1 and complex matches. These three methods are only efficient for centralized environment.

More recently, the database community has begun to exploit P2P technologies for database applications [2, 6, 8, 9, 13, 15, 22, 23, 26]. In [8], the problem of data placement for P2P system was addressed and how data management could be applied to P2P was presented. In [26], the class of “hybrid” P2P systems, where some functionality is still centralized, was studied. In [13], caching of OLAP queries was addressed in the context of a P2P network. Ng et al. [18, 19] and Ooi et al. [20] introduced an IR technique into schema mapping in PDMS. Halevy et al. addressed the issue of schema mediation and proposed a language for mediating between peer schemas in [9]. Hyperion project was proposed in [2, 15, 22], which created schema mapping via mapping tables and required human input. The coDB P2P DB prototype system that measures the performance of various networks arranged in different topologies was proposed in [6].

Schema mapping of existing studies mostly requires human input or intervention. For example, in PeerDB [19], users are expected to provide additional descriptions for the relation and attribute names. In this paper, we would like to take schema mapping one step further by not relying on the additional input imposed on the users. Accordingly, we propose a practical and adaptive solution based on classification and probing.

3 Classification-Based Schema Mapping

In this section, we first give an overview on how to construct a classification scheme for various peer schemas in Section 3.1. Then we describe the classification-based schema mapping in detail in Section 3.2 and Section 3.3.

3.1 Classification Overview

Figure 1 shows the overall architecture of our classification-based schema mapping method. Similar to a conceptual taxonomy, all the shared schemas in our system (relations and their attributes) are classified into certain categories

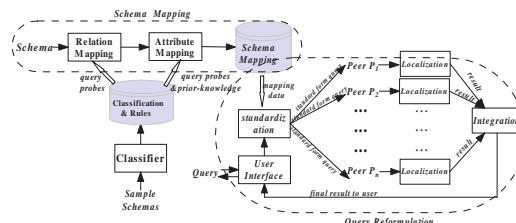


Fig. 1. Architecture of schema mapping based on classification and probing

and each category may contain some subcategories. A hierarchical classification scheme is introduced as follows.

Hierarchical Classification Scheme: A hierarchical classification scheme is a rooted directed tree whose nodes correspond to categories. Category includes relation category and attribute category, and each relation category has some attribute subcategories. An edge from relation category u to another relation category v denotes specialization; while an edge from relation category v to its attribute category v_i denotes the relation w.r.t. v has an attribute w.r.t. v_i .

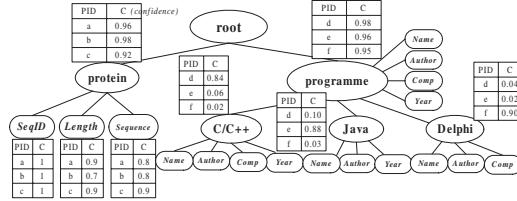


Fig. 2. A classification structure

Figure 2 illustrates a hierarchical classification scheme of our running example, where ellipse denotes relation category and rounded rectangle denotes attribute category. A relation category has several attribute subcategories, which correspond to its attributes. In Figure 2, the *root* node has two relation categories *Protein* and *Programme*, while *Programme* has three relation subcategories - *Java,C/C++,Delphi* and four attribute subcategories - *Name,Author,Comp,Year*.

We have mentioned that the classification structure is maintained by super peers that can be organized with existing overlays, e.g. BATON [12]. Each super peer maintains a subset of categories, where each category is associated with its own classification rules (including prior-knowledge for attribute categories, which will be introduced in Section 3.3), and the physical addresses of the peers that have classified some relations into it. The categories on super peers are indexed with BATON's distributed index facility, so that we can go through the classification hierarchy from any of the super peers.

The local schema mappings are not maintained by super peers but by normal peers. Each peer maintains its local schema mapping with its matched categories, and the identifiers of the relevant peers that have classified schemas into the same categories. With our running example, category *protein* has three peers *a, b, and c* that have classified their schemas into it. Correspondingly, each of peers *a, b* and *c* maintains its local schema mappings with *protein* as shown in Figure 3.

Initially, the hierarchical classification scheme can be extracted from existing classifications using special-purpose languages and tools, or it can also be constructed from scratch. If there are certain sample schemas in PDMS, we can use many existing methods such as naive Bayes classifier [5], C4.5 [21], RIPPER [3], and Support Vector Machine [24], to classify them. On the other hand, if there are no sample schemas, we can construct the classification from scratch: for a new schema, once it matches certain categories in the existing hierarchical

<i>PeerID</i>	<i>Relevant peers</i>	<i>Local Schema</i>	<i>Mapped Category</i>
<i>Peer a</i>	<i>Peer b,c</i>	<u>Kinases</u>	<u>Protein</u>
		<u>ID</u>	<u>SeqID</u>
		<u>len</u>	<u>Length</u>
		<u>seq</u>	<u>Sequence</u>
<i>Peer b</i>	<i>Peer a,c</i>	<u>annexin</u>	<u>Protein</u>
		<u>identifier</u>	<u>SeqID</u>
		<u>length</u>	<u>Length</u>
		<u>seqs</u>	<u>Sequence</u>
<i>Peer c</i>	<i>Peer a,b</i>	<u>protein</u>	<u>Protein</u>
		<u>number</u>	<u>SeqID</u>
		<u>seqlength</u>	<u>Length</u>
		<u>sequence</u>	<u>Protein</u>
		<u>number</u>	<u>SeqID</u>
		<u>seq</u>	<u>Sequence</u>

Fig. 3. Local schema mapping

classification schema, it will be classified into these categories; otherwise, it will be inserted into the hierarchical classification scheme as a new category (which may be constructed as a parent category of some existing categories).

Generally, schema mapping based on classification, in our approach, operates in two phases: 1) Relation mapping (Section 3.2); and 2) Attribute mapping (Section 3.3), which are presented in the following subsections respectively.

3.2 Relation Mapping

We create relation mapping between relevant peers by classifying their relations into the most relevant categories through query probing. The probe-based method has been used for mining hidden-web data in [7][125], which is orthogonal to the schema mapping of our work. Since the classification rules can capture the characteristics of various categories, we generate query probes according to these classification rules, which are used to differentiate various relations. Basically, if a query probe returns expected results from a relation, this relation is related with the category w.r.t. the query probe, and subsequently it is classified into the category.

Now we describe the class of rule-based classifiers and show how we can use a rule-based classifier to generate a set of query probes that will help us estimate the number of results for each category of interest in a relation. In a rule-based classifier, the classification decisions are based on a set of logical classification rules, $\kappa_i \rightarrow C_i$, where the antecedents of the rules are conjunctions of words and the consequents are the category assignments. For example, the following classification rules are part of a classifier for the categories “Java book” and “protein”, respectively.

$$\text{Java AND book} \rightarrow \text{“Java book”}; \quad \%a\%c\%g\%t\% \rightarrow \text{“protein”}$$

Such rules can be used to classify previously unseen relations. For example, the first rule will classify the relation containing the keywords “Java” and “book” into the category *Java book*. The second will classify the relation containing the keywords “%a%c%g%t%” into the category *protein*. We can simulate the behavior of a rule-based classifier over all categories of the classification scheme

by mapping each rule $\kappa_i \rightarrow C_i$ of the classifier into a boolean query q_i that is the conjunction of all keywords appeared in κ_i . Thus, if we send the query probe q_i to a new relation R , the query will match exactly $f(q_i)$ results in R that would have been classified by the associated rule into category C_i . Actually, instead of retrieving the concrete results, we only need keep the number of matches reported for each query probe, and use this number as the measure of whether the probed relation satisfies the corresponding classification rule.

Having the result for each query probe, we can construct a good approximation of the *Weight* and *Confidence* vectors for a relation R . We approximate the number of results of R in category C_i as the total number of matches from all query probes derived from rules with category C_i . Using this information we generate the approximated weight and confidence vectors for R , with which we decide how to classify R into one or more categories in the classification scheme.

Weight vector: Consider a relation R and a hierarchical classification scheme $C=\{C_1, C_2, \dots, C_n\}$, where each category $C_i \in C$ is associated with a classification rule $\kappa_i \rightarrow C_i$. $f(R, \kappa_i)$ represents the number of results when using κ_i to probe R . The weight of relation R for C_i , $\mathcal{W}(R; C_i) = f(R, \kappa_i)$, is the number of answers in R on category C_i .

Confidence vector: In the same setting as weight vector, the estimated confidence of R for C_i , $\mathcal{S}(R; C_i)$, is:

$$\mathcal{S}(R; C_i) = \frac{\mathcal{S}(R; \text{Parent}(C_i)) * \mathcal{W}(R; C_i)}{\sum_{C_j \text{ is a child of Parent}(C_i)} \mathcal{W}(R; C_j)}.$$

As a special case, $\mathcal{S}(R; \text{"root"})=1$.

$\mathcal{W}(R; C_i)$ defines the absolute amount of the results that relation R contains about category C_i , while $\mathcal{S}(R; C_i)$ defines the relative amount of the results that relation R contain about C_i .

As described above, a weight-based classification would classify a relation into a category when the relation has a substantial number of results in the given category. Alternatively, a confidence-based classification would classify a relation into a category when a significant fraction of the results it contains are of this specific category. In general, however, we are interested in balancing both weight and confidence with two associated thresholds, τ_w and τ_c , respectively, as captured in the following. Formally, to classify R into certain categories, we use classification criterion described in the following.

Classification Criterion: Consider a classification scheme C with categories $\{C_1; C_2; \dots; C_n\}$ and a relation R . R is classified into category C_i if it satisfies all the following conditions:

- $\mathcal{W}(R; C_i) \geq \tau_w$, $\mathcal{S}(R; C_i) \geq \tau_c$.
- $\mathcal{W}(R; C_j) \geq \tau_w$, $\mathcal{S}(R; C_j) \geq \tau_c$ for any ancestor C_j of C_i .
- $\mathcal{W}(R; C_k) < \tau_w$ or $\mathcal{S}(R; C_k) < \tau_c$ for any child C_k of C_i .

where $0 \leq \tau_c < 1$ and $\tau_w \geq 1$ are the given thresholds.

With our hierarchical classification scheme, we classify the relations in a top-down way. A new relation is first classified by the root-level classifier and then recursively “pushed down” to the lower level classifiers. A relation R is pushed down to the category C_j when $\mathcal{W}(R; C_j)$ and $\mathcal{S}(R; C_j)$ are no less than thresholds τ_w (for weight) and τ_c (for confidence), respectively. If a category C_k can not match with R , we can prune the whole subtree rooted at C_k . The final set of categories, into which we classify R , is the approximate categories of R in C .

The probe-based method relies on category classifiers to define query probes and obtain category match information for a relation. Unfortunately, classifiers are not always perfect — sometimes they can wrongly classify relations into incorrect categories and leave some relations that do not match any rules unclassified. Here, we present a novel method to adjust the initial probing results in order to avoid such potential errors. It is a common practice in the machine learning community to report classification using a confusion matrix [16]. We adapt this notion for use in our probing scenario.

Confusion Matrix: Consider a classification scheme with categories $\{C_1; C_2; \dots; C_n\}$, for each category C_i , there is a relevant relation R_i mapped with it. Confusion Matrix $\mathcal{M}=(m_{ij})$ is an $n*n$ matrix, where m_{ij} is the number of matches generated from R_j for query probe w.r.t. category C_i , divided by the number of results in R_j .

In a perfect setting, the probes for C_i match only results in R_i and each result in R_i matches exactly one probe for C_i . In this case the confusion matrix is the identity matrix. The process to create confusion matrix is:

- i) Generate the query probes from classification rules of the categories and probe the relations w.r.t. the categories in the classification scheme.
- ii) Create an auxiliary confusion matrix $\mathcal{X}=(x_{ij})$ and set x_{ij} equal to the number of matches from R_j for query probe w.r.t. category C_i .
- iii) Normalize the columns of \mathcal{X} by dividing column j with the number of results in R_j . The result is confusion matrix \mathcal{M} .

Example 1. Suppose that we have a classifier for three categories $C_1 = "C/C++"$, $C_2 = "JAVA"$, $C_3 = "Delphi"$, and there are three relations R_1, R_2, R_3 with 2000, 1500, 1000 records for “C/C++”, “JAVA”, “Delphi”, respectively. After probing these three relations with the three query probes generated from the classification rules, we construct the following confusion matrix. Element $m_{13} = \frac{100}{2000}$ means that it misclassifies 100 records of R_1 into R_3 .

$$\mathcal{M} = \left\{ \begin{array}{ccc} \frac{1600}{2000} & \frac{150}{1500} & \frac{80}{1000} \\ \frac{200}{2000} & \frac{1200}{1500} & \frac{20}{1000} \\ \frac{100}{2000} & \frac{60}{1500} & \frac{840}{1000} \end{array} \right\} = \left\{ \begin{array}{ccc} 0.8 & 0.1 & 0.08 \\ 0.1 & 0.8 & 0.02 \\ 0.05 & 0.04 & 0.84 \end{array} \right\}$$

Interestingly, multiplying the confusion matrix with the weight vector that represents the exact correct number of results for each category, yields, the weight vector with the number of results in each category as matched by the query probes. For instance, in Example 1, there are exactly 2000 results for C_1 , 1500 results for C_2 and 1000 results for C_3 , and the number of probe results are

respectively 1830, 1420, 1000. We can infer the exact weight vector, \mathcal{EW} , from probe result and matrix \mathcal{M} , where $\mathcal{EW}(C) = \mathcal{M}^{-1} * \mathcal{W}(C)$. Hence, when classifying a relation, we will multiply \mathcal{M}^{-1} with $\mathcal{W}(C)$ to obtain a better approximation of the weight vector.

3.3 Attribute Mapping

After classifying a relation into certain relation categories, we have to classify its attributes into the associated attribute categories, which can be performed similarly with relation mapping described in Section 3.2. In addition, since attributes have their own characteristics, we introduce some techniques to improve the accuracy of attribute mapping in this section.

Each attribute of a relation is associated with a particular type, such as string, number, date, etc., and different types capture different characteristics. Moreover, an attribute may be restricted with certain domain, such as attribute Age (age of human beings), is any number between 0 and 150, since there is no person whose age is larger than 150 or less than 0. Therefore, we introduce prior-knowledge for attribute mapping.

Prior-knowledge: Consider relation category C with attribute categories $\{\mathcal{L}_1; \mathcal{L}_2; \dots; \mathcal{L}_p\}$. Each attribute category \mathcal{L}_i must satisfy the prior-knowledge χ_i , represented as: $\mathcal{L}_i \models \chi_i$.

χ_i can be generated manually or automatically, and we generate it automatically based on machine learning technique. Any attribute that does not satisfy χ_i , cannot be mapped with \mathcal{L}_i . Therefore, we can generate a query probe, which does not satisfy χ_i , to probe an unknown attribute \mathcal{A}_j . If there are some results returned for this query probe, it is obvious that the attribute category \mathcal{L}_i cannot map to \mathcal{A}_j ; otherwise, we probe \mathcal{A}_j with the query probe that is generated by the classification rules w.r.t \mathcal{L}_i , and approximate the count of the probing results to represent the correlation of the two attributes. Accordingly, we can more accurately create attribute mapping with the help of the prior-knowledge.

Example 2. Consider category Person(ID, Name, Age, Sex) with prior-knowledge: 1) ID|=Number(0,10000); 2) Name|=String; 3) Age|=Number[0, 150]; 4) Sex|= {alternative of two values(e.g.male,female)}. A relation People has been mapped to the relation category Person. Now we consider how to create attribute mapping between them. Since there are only two values of Sex, we probe each attribute of People through the query probe generated according to the prior-knowledge of Sex: Select count(distinct probe-attribute) from People. If the result of this query probe is larger than two, we can make sure this probe-attribute cannot be mapped to Sex. Also, we probe each attribute of People through the query probe generated according to the prior-knowledge of Age: Select probe-attribute from People where probe-attribute>150 or probe-attribute<0. If the probe query does not return empty, we make sure that probe-attribute cannot match with Age.

Formally, we introduce correlative matrix to create attribute mapping.

Correlative Matrix: Consider category C with attribute subcategories $\{\mathcal{L}_1; \mathcal{L}_2; \dots; \mathcal{L}_p\}$, and each \mathcal{L}_i is associated with a prior-knowledge χ_i . Relation R with

attributes $\{\mathcal{A}_1; \mathcal{A}_2; \dots; \mathcal{A}_q\}$ is relation mapping with \mathcal{C} . The correlative matrix $\text{Corr}(\mathcal{C}, \mathcal{R}) = \{m_{ij}\}$ is a p^*q matrix. $m_{ij}=0$, if \mathcal{A}_j does not satisfy χ_i ; otherwise m_{ij} is the number of results using the query probe generated by the classification rule of \mathcal{L}_i , to probe \mathcal{A}_j .

Relative Correlative Matrix: In the same setting as correlative matrix, the relative correlative matrix $R\text{Corr}(\mathcal{C}, \mathcal{R}) = \{r_{ij}\}$ is a p^*q matrix, and $r_{ij} = \frac{m_{ij}}{\sum_{k=1}^p m_{kj}}$.

Attribute Mapping Criterion: Consider category \mathcal{C} with attribute subcategories $\{\mathcal{L}_1; \mathcal{L}_2; \dots; \mathcal{L}_p\}$ and relation \mathcal{R} with attributes $\{\mathcal{A}_1; \mathcal{A}_2; \dots; \mathcal{A}_q\}$. \mathcal{L}_i maps to \mathcal{A}_j if $\text{Corr}(\mathcal{C}, \mathcal{R}) = \{m_{ij}\}$ and $R\text{Corr}(\mathcal{C}, \mathcal{R}) = \{r_{ij}\}$ satisfy: $r_{ij} \geq \tau_c$ and $m_{ij} \geq \tau_w$, where τ_w, τ_c are thresholds of weight and confidence respectively.

Example 3. Consider category $\mathcal{C} = \underline{\text{Person}}$ with attribute subcategories: $ID, Name, Age, Sex$; Relation $\mathcal{R} = \underline{\text{people}}$ ($p_id, p_name, p_age, gender$). \mathcal{R} is relation mapped to \mathcal{C} , and we demonstrate how to create attribute mapping between \mathcal{C} and \mathcal{R} . We probe each attribute of \mathcal{R} using the prior-knowledge of attribute subcategories in \mathcal{C} (the prior-knowledge in example 2) and get the correlative and relative correlative matrixes. We can see each attribute of \mathcal{R} exactly maps the corresponding attribute of \mathcal{C} with the help of prior-knowledge.

$$\text{Corr}(\mathcal{C}, \mathcal{R}) = \left\{ \begin{array}{c|cccc} & p_id & p_name & p_age & gender \\ \hline ID & 1 & 0 & 0 & 0 \\ Name & 0 & 20 & 0 & 0 \\ Age & 0 & 0 & 2000 & 0 \\ Sex & 0 & 0 & 0 & 2 \end{array} \right\} \quad R\text{Corr}(\mathcal{C}, \mathcal{R}) = \left\{ \begin{array}{c|ccccc} & p_id & p_name & p_age & gender & \\ \hline ID & 1 & 0 & 0 & 0 \\ Name & 0 & 1 & 0 & 0 \\ Age & 0 & 0 & 1 & 0 \\ Sex & 0 & 0 & 0 & 1 \end{array} \right\}$$

4 Reformulation

With the created schema mapping, we can reformulate the query issued to a peer over its peer schema to the queries over the peer schemas of its relevant peers, such that they can understand and answer it. We first define the *standard form query* and *local form query* in our system.

Standard form query: A standard form query is the query composed of relations and attributes of the relational categories and attribute categories in the hierarchical classification scheme.

Local form query: A local form query of peer P is the query composed of the relations and attributes of P 's local peer schema.

Query reformulation with our method operates in three phases, which are described in the following subsections separately.

4.1 Standardization

In the standardization phase, the peer needs to transform the received query into the standard form query, which is represented by certain relation categories and their corresponding attribute categories.

Consider the issued query is represented as a triple $\mathcal{Q} = \langle \mathcal{R}, \mathcal{A}, \mathcal{C} \rangle$, where \mathcal{R} is a relation name, \mathcal{A} is the attribute set composed of $\{\mathcal{A}_1; \mathcal{A}_2; \dots; \mathcal{A}_p\}$, \mathcal{C} is the

condition set (If the query contains more than one relation, we can decompose it into multiple queries with single relation and then integrate them.).

We first find all the categories, $\{\mathfrak{R}_1; \mathfrak{R}_2; \dots; \mathfrak{R}_n\}$, where each \mathfrak{R}_i is mapped to \mathcal{R} , through the local schema mapping. Then we look at the attribute sub-categories of \mathfrak{R}_i , $\mathcal{N}_i = \{\mathcal{N}_{i_1}; \mathcal{N}_{i_2}; \dots; \mathcal{N}_{i_p}\}$, where \mathcal{N}_{i_k} is mapped with \mathcal{A}_k . Let $\text{relevantPeers}(\mathfrak{R}_i)$ and $\text{relevantPeers}(\mathcal{N}_{i_k})$ denote the sets of peers that have classified some relations and attributes in \mathfrak{R}_i and \mathcal{N}_{i_k} , respectively. If $\mathcal{P}_i = \text{relevantPeers}(\mathfrak{R}_i) \cap (\bigcap_{k=1}^p \text{relevantPeers}(\mathcal{N}_{i_k})) \neq \emptyset$, \mathfrak{R}_i has a schema mapping with \mathcal{R} , and we can reformulate \mathcal{Q} to \mathcal{Q}_i by replacing \mathcal{R} with \mathfrak{R}_i and \mathcal{A}_k with \mathcal{N}_{i_k} , and send the standard form query \mathcal{Q}_i to the peers in \mathcal{P}_i . In addition, we can get the set of all the peers, $\mathcal{P} = \bigcup_{i=1}^n \mathcal{P}_i$, which have relations mapped to \mathcal{R} .

4.2 Localization

When the relevant peers receive the standard form query from the query initiator, they need to reformulate it into their local form query over their own peer schemas in order to execute it.

The reformulation process for transforming a standard form query into a local form query is similar to the way described in Section 4.1. We also consider the standard form query as a triple $\mathcal{Q} = \langle \mathcal{R}, \mathcal{A}, \mathcal{C} \rangle$. We first find the set, \mathcal{S} , composed of local relations \mathcal{S}_i that map to \mathcal{R} . If \mathcal{S}_i contains all the attributes in \mathcal{A} , we rewrite \mathcal{Q} by replacing \mathcal{R} with \mathcal{S}_i , and \mathcal{A} with corresponding attributes in \mathcal{S}_i . In some cases, the local peer cannot reformulate the standard form query \mathcal{Q} into a local form query with one relation, because it needs to join several relations to answer \mathcal{Q} . For example, if $\exists \mathcal{S}_i \in \mathcal{S}$ and there is an attribute $\mathcal{A}_k \in \mathcal{A}$, which is not an attribute of \mathcal{S}_i , in this way, there must $\exists \mathcal{S}_j \in \mathcal{S}$ that has an attribute \mathcal{A}_k . If $\mathcal{A} \subseteq \mathcal{S}_i \cup \mathcal{S}_j$, we can answer \mathcal{Q} through joining \mathcal{S}_i and \mathcal{S}_j ; otherwise we need to further find more relation(s) to join with \mathcal{S}_i and \mathcal{S}_j in order to answer \mathcal{Q} .

After a relevant peer answers the reformulated local form query, it returns the results that are encapsulated by the attributes in \mathcal{A} , such that the query initiator can recognize them.

4.3 Integration

When receiving the answers from relevant peers, the query initiator transforms those answers from various peers represented by attributes of the standard form query into the answers represented with its local attributes, and integrates these results to return to the user.

Consider the issued query is in a triple $\mathcal{Q} = \langle \mathcal{R}, \mathcal{A}, \mathcal{C} \rangle$, and its corresponding reformulated standard form queries are represented as $\mathcal{Q}_1 < \mathcal{R}_1, \mathcal{A}_1, \mathcal{C}_1 >; \mathcal{Q}_2 < \mathcal{R}_2, \mathcal{A}_2, \mathcal{C}_2 >; \dots; \mathcal{Q}_n < \mathcal{R}_n, \mathcal{A}_n, \mathcal{C}_n >$. The mapping from the issued query to the standard form queries is one-to-many, but the mapping in reverse is one-to-one. Therefore, the transformation of the attributes in answers is much easier than query reformulation. Suppose a relevant peer returns its results of $\mathcal{Q}_i < \mathcal{R}_i, \mathcal{A}_i, \mathcal{C}_i >$, and since the querying peer has known the mapping from \mathcal{Q} to \mathcal{Q}_i in standardization phase, it can simply transform the attributes in the answers by

Table 1. Data sources in our experiments

Dataset	Schema	# Relations	# Attributes
Amalgam	S_1	15	99
Amalgam	S_2	27	53
Amalgam	S_3	5	27
Amalgam	S_4	9	40
THALIA	S_5	35	242

replacing the attributes in \mathcal{A}_i with the corresponding attributes in \mathcal{A} . Finally, it integrates the results from all relevant peers and returns them to the user.

5 Experimental Study

In this section, we report performance study for evaluating our schema mapping method. The proposed method was implemented in Java. We used the Amalgam schema and data integration test suite [17] and THALIA benchmark [10] as our experimental data sources. Table 1 shows the statistics of the datasets¹. We evaluate our method from two aspects. First, we study the effectiveness of our schema mapping strategy for matching two schemas. Second, we look at the performance of schema mapping and query processing in a real P2P network setting.

5.1 Mapping Between Two Schemas

We first evaluate the quality of mappings obtained with our method between two schemas in this section. Given two schemas, we first classify the relations and attributes of either one schema and get the corresponding classification rules, then we create schema mapping between them by classifying the relations and attributes of the other schema with our probing strategy. We use precision and recall to evaluate the quality of the obtained mappings. Precision is the fraction of the number of correct relation mappings (Correct relation mapping means both relations and their attributes are mapped correctly) and the number of total obtained relation mapping. Recall is the fraction of the number of correct relation mappings obtained and the number of total correct relation mappings.

Consider two schemas S and T , we denote the precision of probing T with S as $P_{\overrightarrow{ST}}$, that is, S is the schema classified firstly. Similarly, $P_{\overrightarrow{TS}}$ is the precision of probing S with T . In addition, we define $\mathcal{F}_{ST} = \mathcal{F}_{TS} = \frac{2 * P_{\overrightarrow{ST}} * P_{\overrightarrow{TS}}}{P_{\overrightarrow{ST}} + P_{\overrightarrow{TS}}}$. These three metrics are used to evaluate the precision of schema mappings between S and T . In the same way, we define corresponding metrics for recall as $R_{\overrightarrow{ST}}$, $R_{\overrightarrow{TS}}$ and $\mathcal{F}'_{ST} = \mathcal{F}'_{TS} = \frac{2 * R_{\overrightarrow{ST}} * R_{\overrightarrow{TS}}}{R_{\overrightarrow{ST}} + R_{\overrightarrow{TS}}}$.

Our method is evaluated in two cases. First, we create schema mappings without prior-knowledge. Second, we create schema mappings with prior-knowledge.

¹ There are 28 databases and 35 tables in THALIA. We transform THALIA data into 35 relation tables, denoted as S_5 , and we also create schema mapping between them.

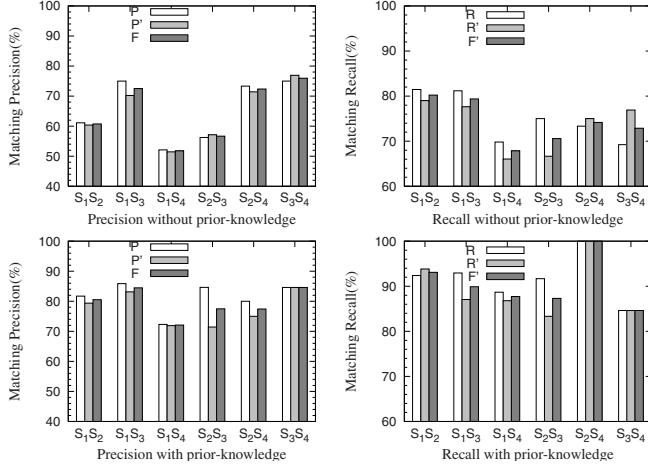
**Fig. 4.** Mapping between two schemas

Figure 4 describes the experimental results of matching 6 pairs of schemas from S_1 to S_4 , where P , P' , R , R' , F and F' denote $P_{\overline{ST}}$, $P_{\overline{TS}}$, $R_{\overline{ST}}$, $R_{\overline{TS}}$, \mathcal{F}_{ST} and \mathcal{F}'_{ST} respectively.

The experiment results show that our method achieves high precision and recall. When there is no prior-knowledge, the precision is about 55-75%, and the recall is about 60-80%. Given some prior-knowledge, the accuracy of schema mapping improves dramatically. The precision reaches 75-90%, and the recall increases to 85-100%. Also, we can see that $P_{\overline{ST}}$ and $P_{\overline{TS}}$ do not make considerable difference, which shows the stability of our method.

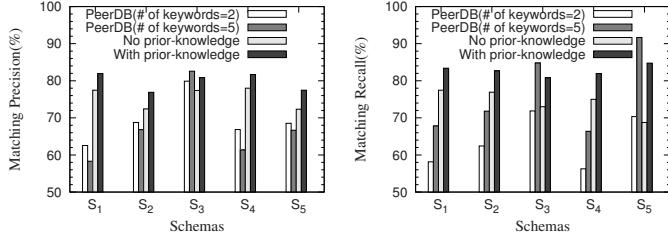
5.2 Mapping in PDMS

In this section, we evaluate our method in PDMS and compare its performance with PeerDB [19]. The experimental environment consists of 32 PCs (thereinto, 8 PCs are super peers) with Intel Pentium 2.4MHz processor and 512M of RAM. All the PCs are running on Windows XP operating system.

We classify some sample schemas using Bayes classifiers [5] into categories, and the eight super peers maintain these categories. Each normal peer shares its peer schema and joins one randomly chosen super peer by classifying its relations into certain categories.

In this experiment, we first evaluate the quality of schema mappings generated with the two approaches, then we compare the effectiveness of query processing in PDMS with the two methods.

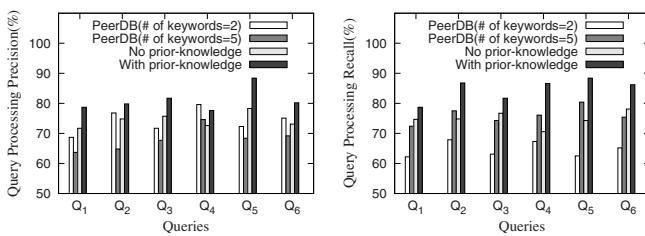
Quality of schema mapping: Similar to Section 5.1, we use precision and recall to evaluate the quality of schema mappings for each of the schema from S_1 to S_5 . Figure 5 shows the experimental results of our method (with and without prior-knowledge) compared with that of peerDB (with 2 keywords annotated for each relation and with 5 keywords annotated for each relation). Not surprisingly,

**Fig. 5.** Schema mapping in PDMS

we can see that in PDMS our schema mapping method with prior-knowledge is more effective than that without prior-knowledge. The precision and recall with prior-knowledge are larger than 80% for most schemas. It can be observed that our method is superior to the PeerDB approach for most schemas (except S_3). Generally, the precision and recall of our method beats that of PeerDB by 10% to 20%. Moreover, PeerDB depends on the keywords annotated to a schema, which must be generated manually. Annotating more keywords to a schema could improve the recall, but degrades the precision. The experiment result shows that our method has good schema mapping performance in PDMS whenever there are overlap instances of the schemas.

Effectiveness of query processing: With the created schema mappings, we evaluate the effectiveness of query processing of the two approaches. We also use the notions of precision and recall for our evaluation. Here precision is defined as the fraction of the number of correct returned answers to the total number of returned answers, and recall is the fraction of the number of correct returned answers to the total number of correct answers.

We generate six queries to evaluate the two methods, in which four queries are based on Amalgam schemas and two are based THALIA schemas. There are two queries that contain join operations. Figure 6 shows the experiment results. Again, we can see that our method is more effective than the PeerDB approach.

**Fig. 6.** Query processing in PDMS

6 Conclusion

In this paper, we propose a method for effective schema mapping based on classification and probing in a PDMS. We classify each peer schema into

certain categories through probing, and the relations in the same category can be mapped to each other. We enhance the classification-based mapping by the application of confusion matrix and prior-knowledge. We also present strategy for reformulating query over a local peer schema to queries on various relevant peer schemas for effective query answering. Our experimented results show that our method achieves high accuracy for schema mapping on real datasets.

Acknowledgement

The work of Guoliang Li and Lizhu Zhou is in part supported by the National Natural Science Foundation of China under Grant No.60573094, the National Grand Fundamental Research 973 Program of China under Grant No.2006CB303103, the National High Technology Development 863 Program of China under Grant No.2006AA01A101, Tsinghua Basic Research Foundation under Grant No. JCqn2005022, and Zhejiang Natural Science Foundation under Grant No. Y105230.

References

1. K. Aberer, P. Cudre-Mauroux, and M. Hauswirth. A framework for semantic gossiping. *SIGMOD Record*, 31(4):505–516, 2002.
2. M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos. The hyperion project:from data integration to data coordination. *SIGMOD Record*, 32(3):53–58, 2003.
3. W. W. Cohen. Learning trees and rules with set-valued features. In *AAAI*, pages 709–716, 1996.
4. R. Dhamankar, Y. Lee, A. Doan, and et al. imap: Discovering complex semantic matches between database schemas. In *SIGMOD*, 2004.
5. R. O. Duda and P. E. Hart. Pattern classification and scene analysis. In *Wiley*, 1973.
6. E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. Queries and updates in the coDB peer to peer database. In *VLDB*, 2004.
7. L. Gravano, P. G. Ipeirotis, and M. Sahami. QProber: A system for automatic classification of hidden-web databases. 21(1):1–41, 2003.
8. S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer. In *WebDB*, 2001.
9. A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *ICDE*, pages 505–516, 2003.
10. J. Hammer, M. Stonebraker, and O. Topsakal. THALIA: Test harness for the assessment of legacy information integration approaches. In *ICDE*, 2005.
11. P. G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: Categorizing hidden-web databases. pages 61–78, 2001.
12. H. V. Jagadish, B. C. Ooi, and Q. H. Vu. BATON: A balanced tree structure for peer-to-peer networks. In *VLDB*, pages 661–672, 2005.
13. P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K. L. Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *SIGMOD*, 2002.

14. J. Kang and J. Naughton. On schema matching with opaque column names and data values. In *SIGMOD*, 2003.
15. A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping data in peer to peer systems: Semantics and algorithmic issues. In *SIGMOD*, 2003.
16. R. Kohavi and F. Provost. Glossary of terms. 30(2/3):271–274, 1998.
17. R. J. Miller, D. Fisla, M. Huang, D. Kymlicka, F. Ku, and V. Lee. Amalgam schema and data integration test suite. www.cs.toronto.edu/~miller/amalgam, 2001.
18. W. S. Ng, B. C. Ooi, and K. L. Tan. Bestpeer: A self-configurable peer-to-peer system. In *ICDE*, 2002.
19. W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. PeerDB:A p2p-based system for distributed data sharing. In *ICDE*, 2003.
20. B. C. Ooi, Y. Shu, and K.-L. Tan. Relational data sharing in peer-based data management systems. *SIGMOD Record*, 32(3):59–64, 2003.
21. J. R. Quinlan. C4.5: Programs for machine learning. In *Morgan Kaufmann Publishers, Inc.*, 1992.
22. P. Rodriguez-Gianolli, M. Garzetti, L. Jiang, and et al. Data sharing in the hyperion peer database system. In *VLDB*, 2005.
23. I. Tatatinov, Z. Ives, J. Madhavan, and A. H. et al. The piazza peer data management project. *SIGMOD Record*, 32(3):47–52, 2003.
24. V. N. Vapnik. Statistical learning theory. In *Wiley-Interscience*, 1996.
25. J. Wang, J.-R. Wen, F. H. Lochovsky, and W.-Y. Ma. Instance-based schema matching for web databases by domain-specific query probing. In *VLDB*, 2004.
26. B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems. In *VLDB*, 2001.
27. C. Yu and L. Popa. Constraint-based XML query rewriting for data integration. In *SIGMOD*, 2004.

ABIDE: A Bid-Based Economic Incentive Model for Enticing Non-cooperative Peers in Mobile-P2P Networks

Anirban Mondal¹, Sanjay Kumar Madria², and Masaru Kitsuregawa¹

¹ Institute of Industrial Science
University of Tokyo, Japan

{anirban,kitsure}@tkl.iis.u-tokyo.ac.jp

² Department of Computer Science
University of Missouri-Rolla, USA
madrias@umr.edu

Abstract. We propose ABIDE, a novel bid-based economic incentive model for enticing non-cooperative mobile peers to provide service in M-P2P networks. The main contributions of ABIDE are three-fold. First, it encourages relay peers to act as brokers for performing value-added routing (i.e., proactively search for query results) due to bid-based incentives. Second, it integrates newly joined peers in the system seamlessly by sharing the loads with the neighbouring brokers. This helps the new peers to earn revenues in order to be able to obtain services. Third, it considers effective data sharing among the peers. ABIDE also considers quality of service, load, energy and network topology. Our performance study indicates that ABIDE is indeed effective in increasing the number of service-providers in M-P2P networks, thereby improving query response times and data availability.

1 Introduction

In a Mobile Ad-hoc Peer-to-Peer (M-P2P) network, mobile peers (MPs) interact with each other in a peer-to-peer (P2P) fashion. Proliferation of mobile devices (e.g., laptops, PDAs, mobile phones) coupled with the ever-increasing popularity of the P2P paradigm  strongly motivate M-P2P network applications. M-P2P application scenarios include a pedestrian issuing a request for an available taxi or a car driver searching for a restaurant nearby his current location. Such P2P interactions among mobile users are generally not freely supported by existing wireless communication infrastructures.

The inherently ephemeral nature of M-P2P environments suggests that *timeliness* of data delivery is of paramount importance in these applications. For example, if a pedestrian looking for an available taxi receives an answer after 20 minutes have already elapsed since he issued the query, he may no longer find the answer to be useful. Furthermore, *data quality* is also a major concern e.g., a mobile user requesting an image could be interested in a high-resolution image.

Incidentally, existing incentive schemes  for M-P2P networks do not address the issue of creating pro-active mobile peers to provide value-added routing service. Moreover, they do not entice the non-cooperative peers in providing service (e.g., providing data to other MPs) to the network by allowing load-sharing so that peers can generate revenues, thereby encouraging seamless participation of peers in the system.

Moreover, the existing schemes in [17][18] deal with data dissemination, while we consider on-demand services. Notably, most peers in P2P systems do not provide any data [5][8][10]. (Nearly 90% of the peers in Gnutella were free-riders [11].) Increased MP participation in providing service to the network would lead to better data availability, likely better data quality, higher available bandwidth and multiple paths to answer a given query. Furthermore, existing schemes do not consider the issue of data quality, which is of considerable importance for M-P2P users.

Given the requirement of timeliness in answering queries, relay MPs should proactively perform *value-added routing* by trying to identify the paths in which the query result could be found quickly and maintain the freshness of the paths. Hence, we propose ABIDE (A BID-based Economic model), which is a novel bid-based incentive model for enticing non-cooperative relay peers to participate in providing service in M-P2P networks. We designate our proposed model as '*ABIDE*' because as we shall see later, every MP benefits in terms of obtaining better service, if it *abides* by the model.

In ABIDE, an MP may provide '*service*' by providing data to other MPs and performing value-added routing by pro-actively searching for targetted peers for query results. Each service in ABIDE is associated with a *price* (in terms of a *virtual currency*). ABIDE requires a data-requesting MP to pay the *price* of the data item to the data-providing MP, thereby encouraging MPs to become data-providers. Data item price depends upon several factors such as access frequency, data quality and estimated response time for accessing the data item. Relay MPs earn a small constant amount of currency for their services.

In our bid-based model, brokers collect bids from data/service providers and then create a summary of recommendation based on the query preferences specified by the users. Based on the bids and the application, users selects a single bid, depending upon the price that a user wants to pay. After a bid is accepted, the requesting peer directly requests the data from the data-providing peer. After the query results have reached the requesting peer, it pays the commission to the broker MP. If a malicious peer avoids paying the commission to the broker MP, the broker MP blacklists it and informs its neighbours regarding the peer's malicious behaviour as a deterrent measure.

In ABIDE, the relay MPs maintain indexes of the services available at other MPs such as data stored at those MPs. The index at different MPs could be different. Using its index, a relay MP can act as a *broker* to pro-actively search for targetted peers for query results. The service-requesting MP needs to pay a *broker's commission* (based on bidding) to the relay MPs, which act as brokers, thereby encouraging them to pro-actively search for query results.(If the relay MP's index does not contain any information concerning the queried service, it selectively forwards the query to its neighbours to earn a relay commission.) Moreover, brokers could cache the paths of frequently queried services, thereby reducing the communication traffic for querying. In the absence of such brokerage, queries would always need to be broadcast (which would flood the network) because there would be little incentive for any MP to cache the paths associated with frequently queried services. Furthermore, a broker MP may also replicate data items that are frequently queried in order to reduce the traffic.

ABIDE also facilitates load-sharing among the MPs as follows. When a broker MP *M* becomes overloaded with too many requests, it transmits its index to relay MPs, who

are willing to store its index. We shall designate such relay MPs as **sub-brokers**. M identifies the sub-brokers by sending a message to its neighbours. Observe that newly joined peers (which are likely to have zero revenue) and existing relay peers would be willing to store the replica of M 's index because it would provide them an opportunity to earn some revenue by performing broker-related functions using M 's index replicated at themselves. Thus, they would be able to actively participate in the network and obtain better service from the network. In essence, the system dynamically creates brokers and sub-brokers based on load and network performance to effectively convert non-cooperative relay MPs into broker MPs.

We define the **revenue** of an MP as the difference between the amount of virtual currency that it earns (by providing services) and the amount that it spends (by requesting services). ABIDE provides an incentive for MPs to provide service to the network so that they can earn more in order to be able to issue their own requests for services. The main contributions of ABIDE are three-fold:

1. It encourages relay peers to act as brokers and sub-brokers for performing value-added routing (i.e., pro-actively search for query results) due to bid-based incentives.
2. It integrates newly joined peers in the system seamlessly by sharing the loads with the neighbouring brokers. This helps the new peers to earn revenues in order to be able to obtain services.
3. It considers effective data sharing among the peers.

ABIDE also considers quality of service, load, energy and network topology. Our performance study indicates that ABIDE indeed increases the number of service-providers in M-P2P networks, thereby improving query response times and data availability.

2 Related Work

Economic models have been discussed in [4][7][12] primarily for resource allocation in distributed systems. A competitive micro-economic auction-based bidding model with support for load-balancing has been proposed in [4]. The proposal in [7] uses game-theoretic and trust-based ideas. The work in [12] examines economy-based optimal file allocation. Incidentally, none of these works address the unique issues associated with the M-P2P environment such as frequent network partitioning and mobile resource constraints. Moreover, they do not address free-riding and incentives for peer participation.

Works concerning free-riding include [5][6][8][10][13][14]. P2P-related free-riding has been discussed in [5]. The works in [6][10][14] propose incentive schemes to combat free-riding. The works in [8] discuss utility functions to capture user contributions, while trust issues are examined in [13]. However, these works do not consider economic models and brokerage to combat free-riding.

Incentive mechanisms for static peer-to-peer networks have been discussed in [15]. However, pre-defined data access structures (e.g., distributed hash tables and searching routing tables), which are used for static P2P networks, are too static in nature to be practically viable for mobile ad-hoc networks. As a single instance, distributed hash tables [16] are not adequate for M-P2P networks because they assume the peers' availability and fixed topology since they are designed for static P2P systems. In essence,

these data access structures have not been designed to handle mobility of peers and frequent network partitioning, which are characteristic of mobile ad-hoc networks. Incentive mechanisms have also been investigated for mobile ad-hoc networks [3][19], the main objective being to encourage a mobile peer in forwarding information to other mobile peers. However, the works in [3][19] do not consider brokerage model, bids and M-P2P architecture. Data replication has been discussed for mobile ad-hoc networks [9], but without considering incentives and prices of data items.

Economic ideas in the context of M-P2P networks have been discussed in [18][17]. While the proposal in [18] addresses issues concerning spatio-temporal data in M-P2P networks, the work in [17] proposes opportunistic dissemination of data in M-P2P networks, the aim being to ensure that the data reaches more people. In contrast, we disseminate data on-demand because transmitting data to MPs, who may not actually require the data, significantly taxes the generally limited energy resources of the MPs. Furthermore, the proposals in [18][17] do not consider brokerage and bidding issues.

3 Data Sharing in ABIDE

Each MP maintains recent read-write logs (including timestamps) of its own data items and the read-logs of the replicas stored at itself. As we shall see shortly, each MP uses this information for computing the prices of the data items and replicas stored at itself. In ABIDE, each data item d is owned by only *one* MP, which can update d *autonomously* anytime; other MPs cannot update d . Memory space of MPs, bandwidth and data item sizes may vary. **Load** $L_{i,j}$ of an MP M_i at time t_j equals $(J_{i,t_j}/B_i)$, where J_{i,t_j} represents the job queue length of M_i at time t_j . Since job queue length is a function of time, load is also a function of time. B_i is the normalized value of the available bandwidth of M_i . $B_i = (B_{M_i} / B_{min})$, where B_{M_i} represents the available bandwidth of M_i and B_{min} is a low bandwidth e.g., we have used $B_{min} = 56$ Kbps.

Each query in ABIDE is a request for a data item. Queries are of the form $(Q_{id}, \tau_S, \tau_H, \epsilon)$, where Q_{id} is the unique identifier of the query, while τ_S and τ_H are the user-specified soft and hard deadlines for answering the query. The significance of ϵ is that the query issuing MP stops accepting bids after ϵ time units have elapsed since the time of query issue (see Section 4). Given that a query Q for a request S is issued at time t_0 , if Q is answered within time $(t_0 + \tau_S)$ (i.e., within the soft deadline), the query issuing MP M_I pays the price μ of S to the query serving MP M_S . However, if Q is answered within the time interval $[t_0 + \tau_S, t_0 + \tau_{HS} + \tau_{HU}]$, M_I pays a reduced price for S to M_S , thereby penalizing M_S for delayed service. As we shall see later, the value of the reduced price depends upon the time delay after the soft deadline τ_S i.e., more delay implies more reduction in price. Finally, if Q is answered after the hard deadline τ_H , M_I does not pay any currency to M_S . Notably, such deadlines for answering queries are necessary due to the inherently ephemeral nature of the M-P2P environment because queries, which are answered after a certain threshold of time has already elapsed, are generally not useful to the user.

In ABIDE, each data item d has a *price* μ (in terms of a *virtual currency*) that quantitatively reflects its relative importance to the M-P2P network. We assume that there could be one original version of d and multiple replicas of d stored at different MPs.

When an MP issues a query for a data item d , it pays the price of d to the MP serving its request. The price μ of d depends upon d 's (recent) access frequency, average query response times (w.r.t. deadlines) for queries on d and data quality of d . An MP M_S computes the price of a data item (or replica) d stored at itself in two steps: (a) M_S first computes the price μ_{rec} of d based on accesses to d during the most recent time period. (We divide time into equal intervals called *periods*, the size of a period being application-dependent.) (b) M_S computes the moving average price μ of d based on the previous N time periods. The moving average price is necessary to take spurious spikes in accesses to d into consideration to ensure that d 's price actually reflects d 's importance. M_S computes μ_{rec} of d as follows:

$$\mu_{rec} = \int_{t_1}^{t_2} \int_0^\delta (\eta dt \times (1/\delta^2) d\delta \times \tau \times DQ \times BA_{M_S} \times PA_{M_S}) / J_{M_S, t_j} \quad (1)$$

where $[t_2 - t_1]$ represents a given time period and δ is the distance between the query issuing MP M_I and the query serving MP M_S (i.e., the MP which stores d and serves the query on d). Given that the positions of M_I and M_S during the time of query issue¹ are (x_I, y_I) and (x_S, y_S) respectively, $\delta = \sqrt{(x_S - x_I)^2 + (y_S - y_I)^2}$ i.e., δ is *Euclidean distance*. Observe how μ_{rec} decreases as δ increases. This is because when the distance between M_I and M_S increases, the response time for queries on d also increases, hence d 's price should decrease. In Equation 1, η is the access frequency of the given data item d during the most recent time period. τ reflects the price reduction (i.e., penalty) due to delayed service. Given that t_0 is the time of query issue, and t_q is the time when the query results reached the query issuing MP, τ is computed as follows.

$$\begin{aligned} \tau &= \mu && \text{if } t_0 \geq t_q \geq (t_0 + \tau_S) \\ &= \mu \times e^{-(t_q - \tau_S)} && \text{if } (t_0 + \tau_S) \geq t_q \geq (t_0 + \tau_S + \tau_H) \\ &= 0 && \text{otherwise} \end{aligned} \quad (2)$$

where τ_S and τ_H are the soft and hard deadlines of a given query respectively. DQ reflects the quality of data provided by M_S for queries on d . DQ is essentially application-dependent. For example, for applications in which image sharing is involved, image resolution would determine data quality. Similarly, for applications in which (replica) consistency is of considerable importance, data quality should be based on data consistency. In general, each MP maintains a copy of the table $T_{\epsilon, DQ}$, which contains the following entries: (x%, high), (y%, medium), (z%, low), where x, y, z are error-bounds, whose values are application-dependent and pre-specified by the system at design time. Essentially, we consider three discrete levels of DQ i.e., *high*, *medium* and *low*, and their values are 1, 0.5 and 0.25 respectively.

In Equation 1, BA_{M_S} is the bandwidth allocated by M_S for d 's download. BA_{M_S} equals $(\sum B_i)/n_d$, where B_i is the bandwidth that M_S allocated for the i^{th} download of d from itself during the most recent time period, while n_d is the number of downloads of d from M_S . As BA_{M_S} increases, μ_{rec} increases because higher bandwidth implies reduced response times for queries on d . PA_{M_S} is the probability of availability of M_S .

¹ We assume that the positions of M_I and M_S do not change significantly between the time of query issue and the time of query retrieval.

When PA_{MS} is high, the implication is that other MPs can rely more on M_S to provide d , hence μ_{rec} increases with increase in PA_{MS} . J_{MS,t_j} is the job queue length at M_S during time t_j . μ_{rec} decreases with increase in the job queue of M_S because when M_S is overloaded with too many requests, M_S 's response time in answering queries on d can be expected to increase.

After computing μ_{rec} , M_S computes the moving average price μ of d . Notably, we use the Exponential Moving Average (EMA), which is capable of reacting quickly to changing access patterns of data items since it gives higher weights to recent access patterns relative to older access patterns. This is consonance with the dynamically changing access patterns that are characteristic of M-P2P networks. M_S computes the price μ of d as follows:

$$\mu = (\mu_{rec} - EMA_{prev}) \times 2/(N + 1) + EMA_{prev} \quad (3)$$

where EMA_{prev} represents the EMA that was computed for the previous time period, and N represents the number of time periods over which the moving average is computed. Our preliminary experiments suggest that $N = 5$ is a reasonably good value for our application scenarios.

An MP M_S earns virtual currency from accesses to its own data items and replicas of others that are stored at itself, and M_S spends currency when it queries for data stored at other MPs. The revenue of an MP M is simply the difference between the amount of virtual currency that M earns and M spends. When an MP joins the M-P2P network for the first time, it has zero currency, hence it first needs to serve other MPs' requests or share some load with neighbouring MPs and in lieu, earn some revenues before it can start issuing its own queries, thereby preventing free-riding. Observe how ABIDE's economy-based paradigm of load-sharing, and replication of data and indexes encourages MPs to increase their revenues, thereby ensuring that they obtain better service from the M-P2P network.

4 Value-Added Routing by Relay MPs in ABIDE

This section discusses value-added routing by relay MPs in ABIDE. Let us henceforth refer to a query issuing MP and a service-providing MP as M_I and M_S respectively.

Basic model of ABIDE: ABIDE provides an incentive to the relay MPs to pro-actively search for the query results as opposed to just forwarding queries. Each MP maintains an index of the services (i.e., data items stored at other MPs.) This index is built by each MP on-the-fly in response to queries that are issued to it. Hence, different MPs have different indexes. An MP M_I issues a query Q using a broadcast mechanism. When any given MP receives the broadcast query, it checks its index. If its index does not contain the identifier of at least one MP that is associated with the query result, it just forwards the query to earn the relay commission. Otherwise, it acts as a broker by issuing a new query for finding the route to locate MPs that can answer the query.

Incidentally, the broker MP's commission is significantly higher than that of the relay MP's commission, which encourages a larger number of non-cooperative relay MPs to index more services, thereby providing them with a higher likelihood of being

able to act as brokers. Broker MPs also cache paths for frequently requested services. Hence, after the system has run for a certain period of time, the need for broadcasting queries can be expected to be significantly reduced. A broker MP may also replicate data items that are frequently queried in order to reduce the querying traffic. A given service-providing MP M_S may also allow a broker MP to store a replica of some of its ‘hot’ data items. In this manner, even if M_S is disconnected, it can still earn revenues. Notably, this also leads to better data availability.

ABIDE also facilitates load-sharing among broker MPs and relay MPs as follows. When a broker MP M becomes overloaded² with too many requests, it sends a message to its neighbours to enquire which of its neighbouring relay MPs would be willing to store a replica of its index. M ’s neighbouring relay MPs, which are willing to store a replica of M ’s index, become the sub-brokers of M . The incentive for these sub-brokers to store a replica of M ’s index is that they would be able to earn revenue by performing broker-related functions using M ’s index replicated at themselves. This would facilitate newly joined MPs and existing relay MPs to seamlessly integrate themselves in the system by actively participating in the network. This effectively converts non-cooperative relay MPs into broker MPs.

Once a given broker MP obtains the route to one or more MPs that can serve the query, it acquires information about the price of the service at each of these MPs. Thus, the broker MP stores information of the form $(S, MP_{id}, \mu, Path)$, where S is the service being requested, MP_{id} is the unique identifier of the MP that can serve the query, and μ is the price of S . $Path$ is simply a linked list data structure containing the list of MPs, which fall in the path between the broker MP and the service-providing MP. In case of multiple paths between the broker MP and the service-providing MP, $Path$ could be a pointer to a set of linked lists (or a two-dimensional array).

Observations concerning the network topology in ABIDE: Suppose a data item d , which exists at multiple MPs albeit possibly with varying quality of data, is being requested as service. Observe that the number of relay nodes between a query issuing MP M_I and a broker MP can vary. Moreover, the number of relay MPs between broker MPs and a given data providing MP M_S can also vary. Thus, the number of hops in the path from M_I to a given service-providing MP M_S can differ. Furthermore, there can be multiple paths from M_I to the same M_S and these paths may pass through multiple brokers. To avoid conflicts among brokers, the broker that occurs first in the traversal starting from M_I would make the bid, while the other brokers in the path would only act as relay MPs.

Interestingly, it is possible for a given $M_S M_a$ to be a one-hop neighbour of M_I . However, some other $M_S M_b$ may be able to provide better data quality and/or lower response time than M_a (e.g., due to low bandwidth between M_b and M_I). Hence, the role of the broker MPs would still be relevant in such cases. In essence, the broker MPs provide M_I with different paths for accessing M_I ’s requested data item d or its replica. This allows M_I to choose the copy of d , which best suits M_I ’s requirements in terms of response time and data quality.

² A broker MP considers itself to be overloaded when its capacity utilization is 60% of its maximum capacity.

Algorithm ABIDE_Query_Issuing_MPs

Inputs: (a) Q : Query (b) d : Queried data item

- (1) Broadcast its query Q for a data item d
 - (2) Receive all the bids that arrive at itself within ϵ time units of issuing the query
 - (3) Evaluate the score γ for each bid
 - (4) Select the bid for which the value of γ is highest and **select** the corresponding broker MP Sel
 - (5) Send message to **selected** broker MP Sel
 - (6) Receive the route to the selected M_S from the broker MP
 - (7) Obtain data item from the selected M_S
 - (8) Send the broker commission to the selected broker MP Sel
- end**

Fig. 1. ABIDE algorithm for Query Issuing MP M_I

Algorithms in ABIDE: Figure 1 depicts the algorithm executed by a query issuing MP, while Figure 2 indicates the algorithm executed by the other MPs, which can either be broker MPs or relay MPs. As Lines 1-2 of Figure 1 indicate, the query issuing MP M_I broadcasts³ its query and waits until ϵ time units have elapsed (since the time of query issuing) to collect the bids from all the brokers. Then M_I determines which bid to accept by computing a score γ , based on the estimated query response time and the data quality (see Line 3). M_I computes γ , where γ equals $(a \times RT + b \times DQ)$. Here, RT and DQ represent the estimated query response time and data quality respectively. The values of RT and DQ are provided to M_I by the broker MP. a and b are weight coefficients which determine the relative weights of RT and DQ , such that $0 \leq a, b \leq 1$ and $a + b = 1$. The values of a and b must be specified by the user because different users have different preferences concerning the relative importance of query response time and data quality essentially due to varying user requirements. DQ is computed in the same manner as discussed for Equation 1. RT equals the data item size divided by the sum of the bandwidths at the intermediate hops between M_S and M_I .

M_I selects the bid with highest value of γ , and selects the broker MP Sel who made that bid (see Line 4). As Lines 5-7 indicate, M_I initiates conversation with selected broker to obtain the query results, which are transmitted from the query serving MP to the query issuing MP via the route suggested by the broker MP.

The algorithm in Figure 2 is executed by MPs, which are either broker MPs or relay MPs. As indicated by Lines 3-14, if the index of a given MP contains the identifier of the queried data item, it acts as a broker, otherwise it just forwards the query. In Line 14, observe that different brokers may bid different amounts of currency for the same data item (or its replica). The amount β of currency that a broker MP bids depends upon the quality of the data item that it is able to provide and the estimated response time for the query issuing MP M_I to receive the data item. Given a data item d of price μ , a given broker MP computes β as $(\mu \times \alpha)$, where α is a percentage of the data item price, hence $0 \leq \alpha \leq 1$. α depends upon the urgency of M_I . Thus, we compute α as e^{-ts} ,

³ After a period of time, if M_I knows a broker MP that can serve the query, broadcast would not be necessary.

Algorithm ABIDE_Brokers_and_Relay_MP

Inputs: (a) Q : Query (b) d : Queried data item

- (1) Receive the broadcast query Q for data item d from query issuing MP M_I
 - (2) Check own index to list the identifier of all the MPs that store d into a set Set_{MS}
 - (3) if Set_{MS} is empty
 - (4) Forward Q to its one-hop neighbours
 - (5) else
 - (6) for each M_S M in Set_{MS}
 - (7) Issue a query to find the route(s) to M
 - (8) List all the routes from itself to M into a set Set_{Route}
 - (9) if Set_{Route} is empty
 - (10) Forward Q to the one-hop neighbours
 - (11) else
 - (12) Select the shortest route R from itself to M based on bandwidths at the intermediate hops
 - (13) Obtain price and data quality information from M
 - (14) Collate all the price, M_S , response time and data quality information with the value of its bid β , and send to M_I
 - (15) Wait for M_I 's reply
 - (16) if M_I accepts bid
 - (17) Obtain identifier of selected M_S from M_I
 - (18) Send a message to selected M_S to send the data item to M_I
 - (19) Receive broker commission from M_I
- end**

Fig. 2. ABIDE algorithm for broker MPs and relay MPs

where τ_S is the soft deadline of the query. Observe that increase in τ_S implies decrease in β due to less urgency.

5 Performance Evaluation

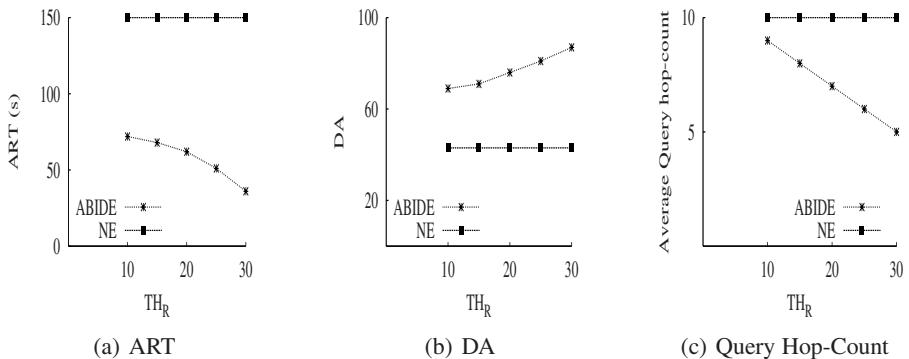
This section discusses our performance evaluation. In our experiments, MPs move according to the *Random Waypoint Model* [2] within a region of area 1000 metres \times 1000 metres. The *Random Waypoint Model* is appropriate for our application scenarios, which involve random movement of users. A total of 200 data items are uniformly distributed among 50 MPs i.e., each MP owns 4 data items. Each query is a request for a data item. In all our experiments, 20 queries/second are issued in the network, the number of queries directed to each MP being determined by the Zipf distribution. Communication range of all MPs is a circle of 100 metre radius. Table I summarizes our performance study parameters.

Our performance metrics are **average response time (ART)** of a query, **data availability (DA)** and **average querying traffic**. ART equals $(1/N_Q) \sum_{i=1}^{N_Q} (T_f - T_i)$, where T_i is the time of query issuing, T_f is time of the query result reaching the query issuing MP, and N_Q is the total number of queries. DA is computed as $((N_S/N_Q) \times 100)$,

Table 1. Performance Study Parameters

Parameter	Default value	Variations
No. of MPs (N_{MP})	50	
Zipf factor (ZF)	0.9	
Queries/second	20	
Bandwidth between MPs	28 Kbps to 100 Kbps	
Probability of MP availability	50% to 85%	
Size of a data item	50 Kb to 350 Kb	
Memory space of each MP	1 MB to 1.5 MB	
Speed of an MP	1 metre/s to 10 metres/s	
Size of message headers	220 bytes	

where N_S is the number of queries that were answered successfully and N_Q is the total number of queries. In ABIDE, queries can fail because MPs, which store queried data items, may be unavailable due to being switched ‘off’ or owing to network partitioning. Average querying traffic is the average number of hops required for query processing in ABIDE. Incidentally, none of the existing proposals for M-P2P networks address economic auction-based revenue models. Hence, as reference, we adapt a non-economic model NE, in which querying occurs by means of the broadcast mechanism. NE does not provide any incentive for the MPs to contribute to the M-P2P network. NE does not perform replication and it does not cache query paths.

**Fig. 3.** Effect of revenue threshold

Effect of variations in the number of MPs above threshold revenue: Threshold revenue TH_R is defined as the ratio of the total revenue of the system to the total number of MPs. In other words, TH_R is the average revenue in the system. Figure 3 depicts the results concerning the effect of variations in the number of MPs above TH_R . The results indicate that when the revenue of more MPs exceed TH_R , ART decreases and data availability increases. This is due to more MPs participating in providing service

as their revenues increase, thereby implying more memory space for holding data items and replicas and more available bandwidth. Moreover, increase in the number of MPs acting as brokers and sub-brokers provide multiple paths for locating a given queried data item. Thus, ABIDE outperforms NE essentially due to the economic incentive nature of ABIDE (which encourages higher MP participation) and load-sharing among brokers and sub-brokers. NE shows relatively constant ART and DA since NE is independent of revenue. The presence of brokers and sub-brokers also reduces the number of hops required for accessing data items because they maintain index of data items and they cache the paths of frequently queried data items, which explains the results in Figure 4.

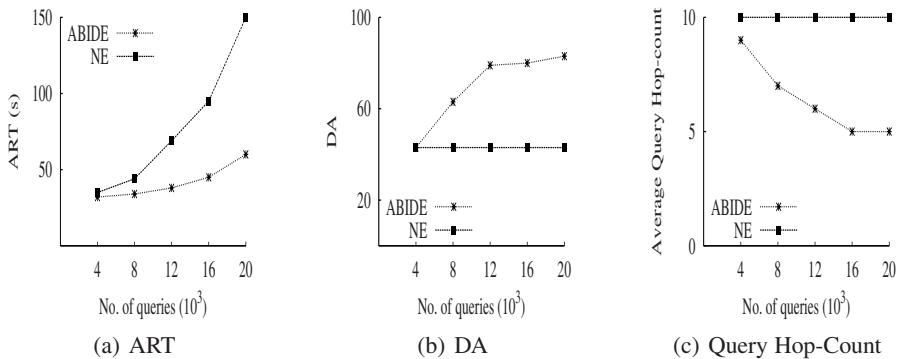


Fig. 4. Performance of ABIDE

Performance of ABIDE: We conducted an experiment using default values of the parameters in Table II. Figure 4a indicates that the ART of both ABIDE and NE increases with time due to the skewed workload ($ZF = 0.9$), which overloads some of the MPs that store ‘hot’ data items, thereby forcing queries to incur high waiting times and consequently high ART. However, over time, more MPs start participating as brokers and sub-brokers in case of ABIDE, thereby providing more memory space and more bandwidth for replication of ‘hot’ data items, which facilitates load-balancing. This explains the increasing performance gap between ABIDE and NE in terms of ART and DA. In Figure 4b, DA eventually plateaus due to reasons such as network partitioning and unavailability of some of the MPs. Furthermore, unlike ABIDE, NE does not maintain the cached routes to the ‘hot’ data items and it does not perform replication, hence ABIDE outperforms NE in terms of query hop-counts. Query hop-counts decrease over time for ABIDE due to replication at the brokers and sub-brokers, and path caching.

6 Conclusion

We have proposed ABIDE, a novel economic bid-based incentive model for enticing non-cooperative mobile peers to provide service in M-P2P networks. ABIDE encourages relay peers to act as brokers for performing value-added routing due to bid-based

incentives, integrates newly joined peers in the system seamlessly by sharing the loads with the neighbouring brokers and considers effective data sharing among the peers.

References

1. E. Adar and B. A. Huberman. Free riding on Gnutella. *Proc. First Monday*, 5(10), 2000.
2. J. Broch, D.A. Maltz, D.B. Johnson, Y.C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocol. *Proc. MOBICOM*, 1998.
3. L. Buttyan and J.P. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *Proc. ACM/Kluwer Mobile Networks and Applications*, 8(5), 2003.
4. D.F. Ferguson, Y. Yemini, and C. Nikolaou. Microeconomic algorithms for load balancing in distributed computer systems. *Proc. ICDCS*, pages 491–499, 1988.
5. M. Fischmann and O. Gunther. Free riders: Fact or fiction? Sep 2003.
6. P. Golle, K.L. Brown, and I. Mironov. Incentives for sharing in peer-to-peer networks. *Proc. Electronic Commerce*, 2001.
7. C. Grothoff. An excess-based economic model for resource allocation in peer-to-peer networks. *Proc. Wirtschaftsinformatik*, 2003.
8. M. Ham and G. Agha. ARA: A robust audit to prevent free-riding in P2P networks. *Proc. P2P*, pages 125–132, 2005.
9. T. Hara and S.K. Madria. Data replication for improving data accessibility in ad hoc networks. *To appear in IEEE Transactions on Mobile Computing*, 2006.
10. S. Kamvar, M. Schlosser, and H. Garcia-Molina. Incentives for combatting free-riding on P2P networks. *Proc. Euro-Par*, 2003.
11. Kazaa. <http://www.kazaa.com/>.
12. J. F. Kurose and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *Proc. IEEE Trans. Computers*, 38(5):705–717, 1989.
13. S. Lee., R. Sherwood, and B. Bhattacharjee. Cooperative peer groups in NICE. *Proc. INFOCOM*, 2003.
14. N. Liebau, V. Darlagiannis, O. Heckmann, and R. Steinmetz. Asymmetric incentives in peer-to-peer systems. *Proc. AMCIS*, 2005.
15. First Workshop on the Economics of P2P Systems.
<http://www.sims.berkeley.edu/research/conferences/p2pecon>. 2003.
16. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Proc. ACM SIGCOMM*, 2001.
17. O. Wolfson, B. Xu, and A.P. Sistla. An economic model for resource exchange in mobile Peer-to-Peer networks. *Proc. SSDBM*, 2004.
18. B. Xu, O. Wolfson, and N. Rishe. Benefit and pricing of spatio-temporal information in Mobile Peer-to-Peer networks. *Proc. HICSS*, 2006.
19. S. Zhong, J. Chen, and Y.R. Yang. Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. *Proc. IEEE INFOCOM*, 2003.

An Efficient Encoding and Labeling for Dynamic XML Data

Jun-Ki Min¹, Jihyun Lee², and Chin-Wan Chung²

¹ Korea University of Education and Technology, Korea
`jkmin@kut.ac.kr`

² Korea Advanced Institute of Science and Technoloy, Korea
`{hyunlee, chungcw}@islab.kaist.ac.kr`

Abstract. In order to efficiently determine structural relationships among XML elements and to avoid re-labeling for updates, much research about labeling schemes has been conducted, recently. However, a harmonic support of efficient query processing and updating has not been achieved. In this paper, we propose an efficient XML encoding and labeling scheme, called EXEL, which is a variant of the region numbering scheme using bit strings. In order to generate the ordinal and insert-friendly bit strings in EXEL, a novel binary encoding method is devised. Also, we devise a labeling scheme for a newly inserted node which incurs no re-labeling of pre-existing labels. These encoding and inserting methods are the bases of efficient query processing and the complete avoidance of re-labeling for updates. Moreover, EXEL supports all structural relationships in XPath and the relationships can be checked by SQL statements supported by an RDBMS. Finally, the experimental results show that EXEL provides fairly reasonable query processing performance while completely avoiding re-labeling for updates.

Keywords: Dynamic XML, Labeling and Update.

1 Introduction

Due to its flexibility and a self-describing nature, XML [2] is considered as the *de facto* standard for data representation and exchange in the Internet. In order to search the irregularly structured XML data, path expressions are commonly used in XML query languages, such as XPath [4] and XQuery [14].

Basically, XML data comprises hierarchically nested collections of elements, where each element is bounded by a start tag and an end tag that describe the semantics of the element. Generally, an XML data is represented as a tree such as DOM [2]. The tree of XML data is implicitly ordered according to the visiting sequence of the depth first traversal of the element nodes. This order is called the *document order*.

Given a tree of XML data, the path information and the structural relationships of nodes should be efficiently evaluated. Diverse approaches such as path index approaches [7, 3] and the reverse arithmetic encoding [10] provide help for obtaining the list of nodes which are reached by a certain path.

In order to facilitate the determination of structural relationships of nodes (e.g., the ancestor-descendent relationship), various labeling methods such as region numbering scheme [17,9] and prefix based scheme [15] have been proposed. In addition, structural modifications to the XML data can occur. For example, insertions of nodes change the structure of a tree of XML data, and the assigned labels may need to be changed. Thus, many researches [15,16,11,8] have been conducted in order to provide an efficient way to handle labels for updating XML data. However, they still cannot entirely remove re-labeling for insertions.

Our Contribution. In this paper, we devise a novel XML encoding and labeling scheme, called EXEL (Efficient XML Encoding and Labeling). EXEL is effective to compute the structural relationships as well as to support the incremental update. The contributions of the paper are as follows:

- **Devise a novel binary encoding:** we devise a novel binary encoding method to generate bit strings which are ordinal and insert-friendly. We extend the region numbering scheme using the bit strings instead of decimal values. The efficient query processing and complete avoidance of re-labeling are based on our binary encoding method.
- **Completely remove re-labeling for updates: we devise a labeling scheme** for a newly inserted node. In our scheme, re-labeling of pre-existing labels for insertion can be completely avoided.
- **Support full axes:** EXEL supports all structural relationships in XPath and the relationships¹ can be checked by SQL statements supported by an RDBMS.

The remainder of the paper is organized as follows. In Section 2, we review various XML labeling schemes. We describe the details of EXEL in Section 3 and present an update method of EXEL in Section 4. Section 5 contains the results of our experiments. Finally, in Section 6, we summarize our work.

2 Related Work

In the region numbering scheme [17,9], each node in a tree of XML data is assigned a region consisting of a pair of start and end values which are determined by the positions of the start tag and the end tag of the node, respectively. Even though all structural relationships represented in XPath can be determined efficiently using $\langle \text{start}, \text{end}, \text{level} \rangle$, an insertion of a node incurs re-labeling of its following and ancestor nodes. [9,11] have tried to solve the re-labeling problem by extending a region and using float-point values. However, the re-labeling problem can not be avoided for frequent insertions after all.

In the prefix labeling scheme [15,5,11], each node in a tree of the XML data has a string label which is the concatenation of the parent’s label and its own identifier. The structural relationships among nodes can be determined by a

¹ In XPath, there are 13 axes. In this paper, we do not consider namespace, and attribute axes since they are not structural relationships.

string function to extract a prefix of a string and string comparison operations. These function and operators degrades a query performance. Dewey labeling scheme [15] and Binary labeling scheme [5] do not require re-labeling for appending leaf nodes. However, they cannot avoid the re-labeling for insertions between two sibling nodes and an insertion of a node between parent and child nodes. Recently proposed ORDPATH [11] is tolerant for insertions. ORDPATH follows a labeling principle similar to the Dewey labeling scheme. In order to avoid re-labeling, it uses only odd numbers for initial labels. When an insertion occurs, it uses an even number between two odd numbers and concatenates an odd number. Although ORDPATH is more bearable for insertions than other approaches, they cannot also avoid re-labeling for an insertion between parent and child nodes.

The prime number labeling scheme [16] uses an inherent feature of the prime number which has only one and itself as its common divisors. The label of a node is a product of its parent node's label and its self-label (i.e., a unique prime number). For the order sensitive query, the prime number labeling scheme uses the simultaneous congruence (SC) values. Even though re-labeling for nodes can be avoided for insertions, the SC values should be re-calculated, and the re-calculation consumes much time. Also, an insertion between parent and child nodes incurs the re-labeling.

In addition, a dynamic quaternary encoding, QED [8], that can be applied to different labeling schemes, has been proposed. In QED, the label size increases by two bits for inserting a node. In contrast, the label size increases by one bit for the insertion in our scheme.

3 Efficient XML Encoding and Labeling (EXEL)

In this section, we present a novel binary encoding method for labeling XML data and an enhanced encoding method to reduce label length. We use the bit strings in the region numbering scheme instead of decimal values for the efficient query processing and the complete elimination of re-labeling for updates.

3.1 Binary Encoding in EXEL

The original region numbering scheme uses decimal values for labels which are sensitive of updates. Therefore, we propose a novel efficient XML encoding and labeling method, called EXEL. It uses bit strings which are ordinal as well as insert-friendly. The bit strings for labeling are generated by the following binary encoding method:

- (1) The first bit string $b(1) = 1$.
- (2) Given the i^{th} bit string $b(i)$, if $b(i)$ contains 0 bit then $b(i+1) = b(i) + 10$. Otherwise, $b(i+1) = b(i)0^k1$, where k is the length of $b(i)$.

Definition 1. Lexicographical order ($<$)

- (i) 0 is lexicographically smaller than 1 ($0 < 1$).
- (ii) if two bit strings a and b are the same($=$), a is lexicographically equal to b .
- (iii) Given bit strings a , b , a' and b' , $ab < a'b'$, if only if $a < a'$ or $a = a'$ and $b < b'$ or $a = a'$ and b is null, where $\text{length}(a) = \text{length}(a')$.

Bit strings generated by the above binary encoding method have the lexicographical orders presented in Definition 1. For example, $1 < 101 < 111 < 1110001$. Also, according to the above generating rule, the bit string always ends with 1. Thus, our encoding scheme satisfies the following property.

Property 1. Given bit strings s_11 and s_21 generated by the above binary encoding method, if $s_11 < s_21$, then $s_1 < s_2$ in the lexicographical order.

Theorem 1 presents the space requirement of our binary encoding scheme.

Theorem 1. In order to encode N ordinal values, the binary encoding of EXEL needs $2^{\lceil \log_2 \log_2 N + 1 \rceil} - 1$ bits, which is about $2 \log_2 N - 1$ bits.

Proof. (i) 1-bit string (i.e., 1) can represent only 1 value. (ii) 3-bit string (i.e., 101, 111) can represent 2 values. (iii) 7-bit string (i.e., 1110001,...,1111111) can represent 2^3 values. (iv) consequently, by the mathematical induction on k , $(2^k - 1)$ -bit string can represent $2^{2^{k-1}-1}$ values.

$$\text{Let } N = 2^0 + 2^1 + \dots + 2^{2^{k-1}-1} = \sum_{i=1}^k 2^{2^{i-1}-1}.$$

By the mathematical induction, $N = \sum_{i=1}^k 2^{2^{i-1}-1} < 2 * 2^{2^{k-1}-1} = 2^{2^{k-1}}$.

Therefore $k = \lceil \log_2 \log_2 N + 1 \rceil$. Consequently, $2^k - 1 = 2^{\lceil \log_2 \log_2 N + 1 \rceil} - 1 \approx 2 \log_2 N - 1$. ■

3.2 Enhancement of Binary Encoding

In the binary encoding method of EXEL, k -bit string has superfluous $(k - 1)/2$ bits consisting of only 1 in order to guarantee the lexicographical order among variable length bit strings. For example, the 7-bit strings from 1110001 to 1111111 can represent only $2^3 = 8$ ordinal values since the first three bits are 111 and the last bit is 1. In order to remove the superfluous part, we devise another binary encoding method with a predefined length of a bit string. The predefined length is obtained from the total number of ordinal values which would be encoded. For labeling a tree of XML data, it is determined by the total number of nodes. The bit string with a predefined length is generated by the following rule:

Let N be the total number of values.

- (1) The first bit string $b(1) = 0^{\log_2 N} 1$.
- (2) Given i^{th} bit string $b(i)$, $b(i+1) = b(i) + 10$.

According to the above rule, a bit string ends with 1 like the original binary encoding method, and it satisfies Property 1.

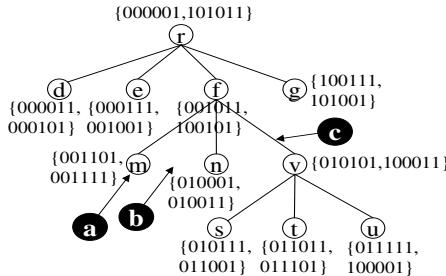


Fig. 1. An example of a tree labeled by EXEL

3.3 Region Labeling in EXEL

In EXEL, we extend the region numbering scheme using ordinal bit strings rather than decimal values. The start and end values are ordinal bit strings generated by the binary encoding of EXEL. Fig 1 shows an example of a tree labeled by EXEL using the binary encoding with a predefined length.

EXEL uses the parent information to determine the parent and child relationships in order to support efficient updates although the original region numbering scheme uses the level information. The use of the parent information also results in an improvement of the query performance. For the simplicity, the parent information (i.e., the start value of the parent's label) is not presented in Fig 1.

The following theorem presents the space requirement of EXEL using the binary encoding with a predefined length.

Theorem 2. *Given the total number of nodes N , the binary encoding using a predefined length in EXEL needs $\log_2 N + 1$ bits for each bit string. Also, the region labeling in EXEL requires $3(\log_2 N + 1)$ for start, end, and parent information.*

3.4 Query Processing

EXEL supports all XPath axes (i.e., ancestor, descendent, parent, child, following, preceding, following-sibling, and preceding-sibling) as the same way in the region numbering scheme because EXEL is based on the original region numbering scheme. For example, in Fig 1, f is an ancestor of t since $s_f (= 001011) < s_t (= 011011)$ and $e_t (= 011101) < e_f (= 100101)$, where (s_x, e_x) is the region of a node x .

For many years, intensive research on storing and managing XML data as the relational data have been conducted. By using an RDBMS, we can utilize the stable repository as well as the efficient query optimizer and the executor. Therefore, we store XML data into relational tables. For a node x in a tree of XML data, the relational table stores its region (s_x, e_x) and its parent's information ps_x . All above conditions for the corresponding axes in the XPath can be expressed by simple SQL statements supported by an RDBMS. For example, the SQL statement to find all descendants of a node f in Fig 1 is `SELECT * FROM NODE WHERE 001011 < start AND end < 100101`.

4 Update

In this section, we present the update behaviors of EXEL. Since the deletion does not incur the re-labeling of nodes, we present the algorithm for the insertion.

4.1 Labeling for Update

The algorithm *MakeNewBitString* makes a new bit string between two pre-existing bit strings. This algorithm can be applied to the original binary encoding and the binary encoding with a predefined length. \oplus denotes the concatenation of two bit strings.

```
Algorithm MakeNewBitString( leftB, rightB )
begin
1.   if length(leftB) > length(rightB) then newB := leftB  $\oplus$  1;
2.   else newB := (rightB with the last bit changed to 0)  $\oplus$  1;
3.   newB := newB  $\oplus$  1;
4.   return newB;
end
```

For example, when we insert two bit strings successively between 101 and 111, the first one is 1101 ($101 < 1101 < 111$) and second one is 11011 ($1101 < 11011 < 111$). For bit strings generated by our binary encoding method, the lexicographical order has a property as follows.

Property 2. Given bit strings s_{11} and s_{21} generated by the binary encoding method of EXEL, if $s_{11} < s_{21}$, then $s_{11} < s_{201}$ and $s_{11} < s_{21}$.

For example, let $s_{11} = 000011$ and $s_{21} = 000101$, then $000011 < 000101$ and $0000111 < 000101$. Through the above property, we can explain that a new bit string generated by the algorithm *MakeNewBitString* preserves the lexicographical order among pre-existing bit strings.

Theorem 3. *The bit string generated by the algorithm *MakeNewBitString* preserves the lexicographical order.*

Proof. If $\text{length}(leftB) > \text{length}(rightB)$, then $leftB < newB (= leftB \oplus 1)$ (by Definition 1) and $newB < rightB$ (by Property2). Otherwise, given $leftB = s_{11}$ and $rightB = s_{21}$, $newB (= s_{20} \oplus 1) < rightB (= s_{21})$ (by Definition 1), and $leftB (= s_{11}) < newB (= s_{201})$ (by Property 2) ■

4.2 Update Processing

There are three kinds of insertions in XML data according to the positions in which nodes are inserted; inserting a child of a leaf node, inserting a sibling and inserting a parent.

```
Algorithm InsertChildOf( cur )
begin
1.    $s_{new} := \text{MakeNewBitString}(s_{cur}, e_{cur});$ 
2.    $e_{new} := \text{MakeNewBitString}(s_{new}, e_{cur});$ 
3.    $ps_{new} := s_{cur};$ 
4.   Insert new node with  $s_{new}$ ,  $e_{new}$ , and  $ps_{new}$ ;
end
```

The algorithm *InsertChildOf* inserts a node as a child of a leaf node *cur*. In EXEL, a region of a child node is contained in that of its parent node. Thus, a region of a inserted node *new*, (s_{new}, e_{new}) should satisfy $s_{cur} < s_{new} < e_{new} < e_{cur}$. The algorithm *MakeNewBitString* is used to make the start and end values of a node *new*. Additionally, the parent information of *new*, ps_{new} will be the start value of *cur* (i.e., s_{cur}). For example, in Fig 11 a node *a* is inserted into a child of a leaf node *m*. The region of *a*, (s_a, e_a) should satisfy $s_m < s_a < e_a < e_m$. Thus, $s_a=0011101$, $e_a=00111011$, and $ps_a=001101$. Algorithms of inserting a sibling and a parent (e.g., in Fig 11, inserting *b* and *c*, respectively) are similar to that of inserting a child. We omit them for want of space.

The insertion of a parent incurs the increase of levels of its all descendants in the original region numbering. However, EXEL keeps the parent information instead of the level. Even if a node is inserted as an ancestor, the parents of the descendent are still unchanged except the child of the inserted node. Consequently, EXEL guarantees the complete avoidance of re-labeling for insertions in any positions even between parent and child nodes.

Even in case of a subtree insertion, the labeling can be efficiently handled. We first apply our labeling method to the subtree. Second, we generate a new bit-string *x* according to the inserting point (p, q) using the algorithm *MakeNewBitString*, then truncate the last bit (i.e., ‘1’) of *x*. Let the truncated bit-string be *x'*. We complete labeling for the subtree by attaching *x'* as a prefix into the labels of the subtree’s nodes. Since the prefixes of subtree’s nodes are equal, lexicographical orders among labels of subtree’s nodes are preserved. Note that, in the lexicographical order, $p \leq x' < q$. Thus, by Definition 1-(iii), labels of subtree’s nodes (whose prefixes are *x'*) are greater than *p* and smaller than *q*. Therefore, the generated labels still keep the lexicographical order among the pre-existing labels.

5 Experiments

We empirically compared the performance of EXEL with the those of region numbering scheme, the prefix labeling schemes (i.e., ORDPATH and QED-PREFIX), and the prime numbering scheme using synthetic data as well as real-life XML data sets.

5.1 Experimental Environment

The experiments were performed on an Intel Pentium 3GHz with 1GB memory, running Window XP. The XML data sets were stored on an RDBMS,

Table 1. XML Data Set

Data	Name	Size(MB)	# of nodes
XMark	XM1	1	33152
	XM50	50	1390697
	XM115	115	3231322
Shakespeare	S7	7.7	328778

Table 2. Database size

Labeling Scheme	DB Size(MB)
EXEL2 (EXEL1)	147(163)
Region Numbering	119
QRDPATH	140
QED-PREFIX	161
Prime	146

Table 3. Query Set

Name	Query Definition	Name	Query Definition
XQ1	//bidder/ancestor::open_auction	SQ1	//SPEAKER/ancestor::ACT
XQ2	//parent//city	SQ2	//ACT//LINE
XQ3	//name/parent::person	SQ3	//SCENE/parent::ACT
XQ4	//person/name	SQ4	//ACT/TITLE
XQ5	//category/following::person	SQ5	//ACT[3]/following::SPEECH
XQ6	//buyer/preceding::category	SQ6	//TITLE/preceding::ACT
XQ7	//person[2]/following-sibling::person	SQ7	//ACT[6]/following-sibling::ACT
XQ8	//item[46]/preceding-sibling::item	SQ8	//SPEECH[40]/preceding-sibling::SPEECH

PostgreSQL 8.1. We have implemented two versions of EXEL; EXEL1 and EXEL2 using the original binary encoding and the binary encoding with a pre-defined length, respectively. The storage spaces according to the encoding methods have been observed. In order to show the efficiency of EXEL for the query processing and the update processing, we compared EXEL with other representative labeling methods; the region numbering scheme, ORDPATH, QED-PREFIX and the prime numbering scheme. In our experiments, QED was applied to prefix labeling scheme, and we call it QED-PREFIX. In database, we store each element name, its label, and its parent’s label according to the labeling schemes. We evaluated EXEL using XMark data [13] and Shakespeare data [6]. The characteristics of the data sets are summarized in Table 1.

The queries used in our experiments are described in Table 3. The first character in a query name indicates the data set on which the query is executed: ‘X’ denotes XMark, and ‘S’ is for Shakespeare. We evaluated the query performance for all axes in XPath. The number in a query name denotes the type of a query according to the axis contained in the query (i.e., 1 for ancestor, 2 for descendent, 3 for parent, 4 for child, 5 for following, 6 for preceding, 7 for following-sibling, and 8 for preceding-sibling). All experiments were repeated 10 times and we used the average of the processing times except the minimum and maximum values.

5.2 Experimental Results

Query Performance. In our experiments, a given XPath query was transformed to the corresponding SQL statement according to labeling schemes, and the SQL statements were executed on a database. A query parsing time and a query translation time are similar and very small for all labeling schemes. Therefore, in this paper, we present only SQL execution time. The query execution time over various sized data sets are shown in Fig 2. The notation NT means NOT MEASURED due to excessive processing time.

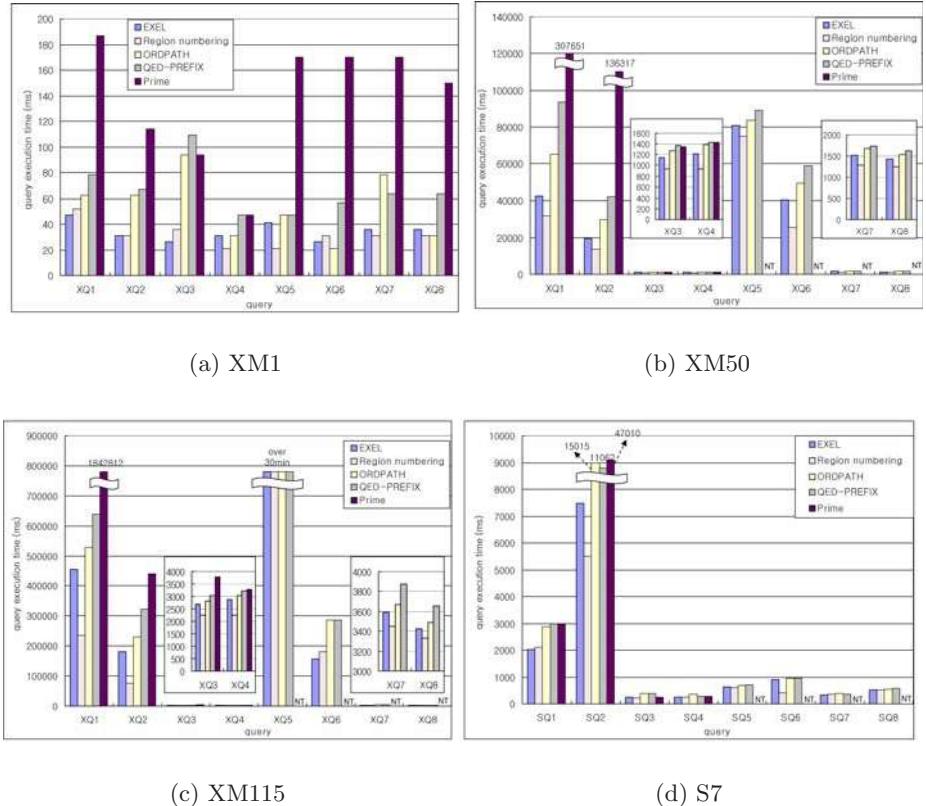


Fig. 2. Query execution time

First of all, ORDPATH and QED-PREFIX should additionally use a string function to extract prefix of a string to compute structural relationships. The use of the string function degrades their performance. In Fig 2(b), the performance of EXEL is about 1.5 times better than that of ORDPATH and 2 times than QED-PREFIX for the query XQ1. As the data size increases, differences of processing time increases.

For queries with ancestor (i.e., XQ1), descendent (i.e., XQ2), following (i.e., X5) and preceding (i.e., XQ6) relationships, the region numbering scheme is better than EXEL since the label size of the region numbering scheme is smaller than that of EXEL. However, the performance of EXEL is close to that of the region numbering scheme compared with other labeling schemes.

Additionally, for finding a parent (i.e., XQ3), children (i.e., XQ4) and siblings (i.e., XQ7 and XQ8), the difference of the query performance among the labeling schemes is not considerable due to the use of the parent information. However, EXEL is still better than ORDPATH and QED-PREFIX.

The prime numbering scheme shows bad performance for ancestor and descendent relationships since it uses the *mod* operation which is more expensive

Table 4. The performance of inserting

Inserting a child						
Labeling Scheme	Data	Time(ms)	Re-labeling	Data	Time(ms)	Re-labeling
Region numbering	XM1	31	0	XM50	16	0
		31	0		32	0
	XM11	31	0	XM115	5	0
		156	11205		15203	468736
EXEL	XM1	15	0	XM115	15	0
		5	0		16	0
	XM11	16	0	XM115	27203	1089457
		3485	109368			
Inserting a sibling						
Labeling Scheme	Data	Time(ms)	Re-labeling	Data	Time(ms)	Re-labeling
Region numbering	XM1	47	0	XM50	688	0
		47	0		922	0
	XM11	63	0	XM115	906	0
		235	17039		21890	717133
EXEL	XM1	187	0	XM115	1469	0
		282	0		3359	0
	XM11	234	0	XM115	4922	0
		5484	167772		38750	1666222
Inserting a parent						
Labeling Scheme	Data	Time(ms)	Re-labeling	Data	Time(ms)	Re-labeling
Region numbering	XM1	31	1	XM50	1656	1
		63	3344		5734	141572
	XM11	63	3344	XM115	6350	141572
		125	11429		19141	469031
EXEL	XM1	453	1	XM115	3703	1
		609	32667		10266	330135
	XM11	834	32667	XM115	13016	330135
		3938	109612		31187	1089687

than the comparison operations for integers and bit strings. For order sensitive queries (i.e., query type 5, 6, 7 and 8), the performance is poorer than those of other labeling schemes since SC-values should be used to compute the document order of a node. Moreover, it takes very long time to compute SC values by an algorithm in [16] even for small data. Thus, we could not measure the query time for the order sensitive queries for over 1MB data.

Consequently, EXEL is superior to the prefix labeling scheme and the prime numbering scheme over all cases. Also, EXEL is comparable with the region numbering scheme than others. This is achieved by the binary encoding scheme generating the ordinal bit strings which can be effectively adopted to the region numbering scheme.

Update Performance. We evaluated the performance of three kinds of insertions; inserting a child node of a leaf node, inserting a next sibling node of a node, and inserting a parent node. In our experiments, we excluded the prime numbering scheme since it requires very expensive re-calculations of SC values even for small data. The influence of inserting a subtree on pre-existing labels is the same as that of inserting a node. Therefore, we omitted the experiment of inserting a subtree. In order to evaluate update performance, we randomly selected a node (a leaf node for inserting a child) for each kind of insertion and inserted a node as its child, next sibling, or parent. For fair comparisons, we

used the same node for all labeling schemes. Table 4 shows the performance of inserting a node.

In the region numbering scheme, the re-labeling was inevitable for all kinds of insertions. For inserting a child to a leaf node, other labeling schemes did not require re-labeling after the insertion. For inserting a next sibling node, in EXEL and ORDPATH, re-labeling of nodes is not incurred. However, in order to generate a label for a newly inserted node, they need to know the label of the next sibling. The performance of EXEL to find the following-sibling of a node is better than those of other labeling schemes, so the time spent to insert a sibling node in EXEL is smaller than those in others. For inserting a node between parent and child nodes, ORDPATH should re-assign labels for the child and its all descendants. EXEL keeps the parent information which is invariant for insertions of ancestors except a parent. Therefore, in EXEL, only one update was incurred. EXEL needs the labels of the previous and next sibling nodes to generate a new label for an inserted node. However, the time to find labels of siblings is much smaller than the time for re-labeling.

In summary, EXEL achieves the complete removal of re-labeling for insertions. Therefore, EXEL can save much time for updates. Since the time measure smaller than 100ms is unstable and less significant, the comparison of execution time over 100ms shows that the update performance of EXEL is 2.3 times on the average and up to 3.8 times better than those of ORDPATH, with the performance gap increasing as the size of XML data gets larger.

Storage Space. Table 2 shows the size of the databases where XM50 is stored using each labeling scheme. EXEL2 reduces the space requirement effectively. EXEL2 requires an additional scan of data to count the number of nodes before labeling. However, the time for the preprocessing is much smaller than the total storing time. Although EXEL2 uses three binary coding values (i.e., start, end, and parent's start), the space requirement is only slightly larger than ORDPATH and Prime numbering scheme. However, the query performance of EXEL is better than them as shown through the experiment results. Moreover, the database size of QED-PREFIX is bigger than EXEL2. EXEL needs a larger space than the region numbering scheme due to the use of the insert-friendly bit string. However, the significant improvement of the update performance according to the use of the bit strings compensates for the space overhead.

6 Conclusion

We propose EXEL, an efficient XML encoding and labeling method which supports efficient query processing and updates. A novel binary encoding method used in EXEL generates ordinal and insert-friendly bit strings. EXEL is a variant of the region numbering scheme using bit strings generated by the novel binary encoding method. EXEL supports all axes in XPath, and the conditions to compute the structural relationships can be simply expressed by SQL statements of an RDBMS. Furthermore, we proposed a labeling method for a newly inserted node, so EXEL removes the re-labeling overhead entirely unlike other

existing labeling schemes. The experimental results show that EXEL provides fairly reasonable query performance. Also, the update performance of EXEL is better than those of existing labeling schemes, with performance gap increasing as the size of XML data gets larger.

Acknowledgements. This research was supported by the Ministry of Information and Communication, Korea, under the College Information Technology Research Center Support Program, grant number IITA-2006-C1090-0603-0031.

References

1. T. Amagasa and M. Yoshikawa. QRS: A Robust Numbering Scheme for XML documents. In *Proc. of ICDE 2003*, pages 705–707, 2003.
2. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation, <http://www.w3.org/TR/REC-xml>, 2004.
3. C.-W. Chung, J.-K. Min, and K.-S. Shim. APEX: An Adaptive Path Index for XML Data. In *Proc. of ACM SIGMOD 2002*, pages 121–132, 2002.
4. J. Clark and S. DeRose. XML Path Language(XPath) Version 1.0. W3C Recommendation, <http://www.w3.org/TR/xpath>, 1999.
5. E. Cohen, H. Kaplan, and T. Milo. Labeling Dynamic XML Trees. In *Proc. of PODS 2002*, pages 271–281, 2002.
6. R. Cover. The XML Cover Pages. <http://www.oasis-open.org/cover/xml.html>, 2001.
7. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. of VLDB 1997*, pages 436–445, 1997.
8. C. Li and T. W. Ling. QED: A Novel Quaternary Encoding to Completely Avoid Re-labeling in XML Updates. In *Proc. of ACM CIKM 2005*, pages 501–508, 2005.
9. Q. Li and B. Moon. Indexing and Querying XML Data for Regular Expressions. In *Proc. of VLDB 2001*, pages 367–370, 2001.
10. J.-K. Min, M.-J. Park, and C.-W. Chung. XPRESS: A Querable Compression for XML Data. In *Proc. of ACM SIGMOD 2003*, pages 122–133, 2003.
11. P. O’Neil, E. O’Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHS: Insert-Friendly XML Node Labels. In *Proc. of ACM SIGMOD 2004*, pages 903–4908, 2004.
12. R. W. Philippe Le Hegaret and L. Wood. XML Path Language(XPath) Version 1.0. <http://www.w3.org/DOM>, 2005.
13. A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proc. of VLDB*, pages 974–985, 2002.
14. D. C. Scott Boag, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML Query Language. W3C Recommendation, <http://www.w3.org/TR/xquery/>, 2005.
15. I. Tatarinov, S. D. Viglas, K. Beyer, J. Shannmugasundaram, E. Shekita, and C. Zhang. Storing and Querying Ordered XML Using a Relational Database System. In *Proc. of ACM SIGMOD 2002*, pages 204–215, 2002.
16. X. Wu, M. L. Lee, and W. Hsu. A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. In *Proc. of ICDE 2004*, pages 66–78, 2004.
17. C. Zhang, J. Naughton, D. Dewitt, Q. Luo, and G. Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *Proc. of ACM SIGMOD 2001*, pages 425–436, 2001.

An Original Semantics to Keyword Queries for XML Using Structural Patterns

Dimitri Theodoratos and Xiaoying Wu

Department of Computer Science,
New Jersey Institute of Technology, USA
`{dth, xw43}@njit.edu`

Abstract. XML is by now the de facto standard for exporting and exchanging data on the web. The need for querying XML data sources whose structure is not fully known to the user and the need to integrate multiple data sources with different tree structures have motivated recently the suggestion of keyword-based techniques for querying XML documents. The semantics adopted by these approaches aims at restricting the answers to meaningful ones. However, these approaches suffer from low precision, while recent ones with improved precision suffer from low recall.

In this paper, we introduce an original approach for assigning semantics to keyword queries for XML documents. We exploit index graphs (a structural summary of data) to extract tree patterns that return meaningful answers. In contrast to previous approaches that operate locally on the data to compute meaningful answers (usually by computing lowest common ancestors), our approach operates globally on index graphs to detect and exploit meaningful tree patterns. We implemented and experimentally evaluated our approach on DBLP-based data sets with irregularities. Its comparison to previous ones shows that it succeeds in finding all the meaningful answers when the others fail (perfect recall). Further, it outperforms approaches with similar recall in excluding meaningless answers (better precision). Since our approach is based on tree-pattern query evaluation, it can be easily implemented on top of an XQuery engine.

1 Introduction

XML is by now the de facto standard for exporting and exchanging data on the web. XML data are represented in a tree structured form.¹ Structured query languages for XML are based on the specification of tree patterns to be matched against the data tree.

The semistructured nature of XML poses problems when it comes to query data on the web using query languages based on tree patterns. First, XML data does not have to comply with a schema and writing a Tree Pattern Query (TPQ) in this context becomes intricate. Second, even if the data complies with some schema, the syntax of a structured language like XQuery is much more complex than a keyword query and therefore, not appropriate for the naive user. Third, a user might not have full knowledge of the schema of the document. Then, formulating a TPQ that retrieves the desired results without

¹ ID/IDREF links would require the modeling of XML documents as graphs but we ignore these features here for simplicity.

being too general can be extremely cumbersome. Finally, data sources usually export data on the web under different structures even if they export the same information. Since elements may be ordered differently in these structures, a single TPQ is not able to retrieve the desired information from all of them.

These issues have been identified early on and attempts have been made to exploit keyword-based techniques used by current search engines on the web for HTML documents. Two main modifications have to be made to these techniques so that they are applicable to XML documents. First, they have to be able to distinguish between values/text (data) and tags/elements (metadata). Second, they have to be able to return fragments of the documents that contain the keywords, as this is appropriate for XML, instead of links to documents. Several approaches suggest keyword-based queries as standalone languages [16][10][5]. Others, extend structured query languages for XML (e.g. XQuery) with keyword search capabilities [6][14].

The problem. Keyword queries usually return to the user a large percentage of XML document fragments that are meaningless (that is, the keywords are matched to unrelated parts of the document). To cope with this problem most approaches assign semantics to queries using some variation of the concept of Lowest Common Ancestor (LCA) of a set of nodes in a tree [16][10][5][19]. However, in most practical cases, the information in the XML tree is incomplete (e.g. optional elements/values in the schema of the document are missing), or irregular (e.g. different structural patterns coexist in the same document). For instance, examining the DBLP data set (data collected in May 2006) we found that almost 10% of the ‘book’ entries and over 1% of ‘article’ entries do not have an author while almost all ‘proceedings’ entries do not have authors (this latter one is reasonable and expected). In such cases, these approaches, even if they succeed to retrieve all the meaningful answers, they comprise only a tiny percentage of meaningful answers in their answer set. Most of the answers are meaningless. In other words, these approaches have low precision. Our experiments in Section 6 with DBLP-based data sets show that in some cases their precision falls below 1% for some approaches. Clearly, such a low precision is a serious drawback for those approaches.

A recent approach [14] introduces the concept of Meaningful Lowest Common Ancestor Structure (MLCAS) for assigning semantics to keyword queries. It also adds new functionality to XQuery to allow users to specify optional structural restrictions on the data selected by the keyword search. The goal is to improve the precision of previous approaches. However, the percentage of meaningful answers returned by this approach (i.e. the recall) is low when the data is incomplete. In our experiments in Section 6, the recall of the MLCAS approach falls below 60% for several cases of incomplete XML data. Clearly, the poor recall cannot be improved by further imposing structural restrictions. This performance is not satisfactory for data integration environments for which this approach is intended.

Our approach. In this paper we suggest an original approach for assigning semantics to keyword queries for XML documents. The originality of our approach relies on the use of structural summaries of the XML document for identifying structural patterns (in the form of TPQs) for a given query. Using a transformation for TPQs we identify those of them (called meaningful TPQs) that return meaningful answers. Previous approaches identify meaningful answers by operating *locally* on the *data* (usually

computing lowest common ancestors of nodes in the XML tree). In contrast, our approach operates *globally* on *structural summaries* of data to compute meaningful TPQs. This overview of data gives an advantage to our approach compared to previous ones.

Contribution. Our main contributions are the following:

- We introduce a simple keyword query language for XML that allows the specification of elements and values of elements (atomic predicates) (Section 3). This language allows the user to specify queries without knowledge of the structure of the XML tree, and without knowledge of a complex TPQ language like XQuery.
- We define structural summaries of data called index graphs. We show how index graphs can be used to compute a set of TPQs for a keyword query that together compute the answer of that query (Section 4).
- Based on a transformation for the TPQs of a query we determine those of them that are meaningful. The meaningful TPQs are used to assign semantics to keyword queries (Section 5).
- Since the meaningful TPQs are tree pattern queries they can be evaluated using any XQuery engine. Therefore, their execution can profit from optimization techniques developed up to now for XQuery (e.g [11][14]).
- We compare our approach with other prominent keyword-based approaches and also with the MLCAS approach. We analyse cases where our approach succeeds in returning meaningful answers that escape other approaches. We also analyse cases where our approach succeeds in excluding meaningless answers that are returned by other approaches.
- We have implemented and experimentally evaluated our approach both on complete and incomplete real XML data. Our approach shows better recall compared to previous ones. In addition, it allows for a better precision among approaches with similar recall (Section 6).

2 Related Work

A number of papers deal with the assignment of meaningful semantics to keyword-based query languages for XML [16][10][5][14][19]. All of them are based on some variation of the concept of Lowest Common Ancestor (LCA). The query language in [5] allows also some primitive structural restrictions to be expressed. The approach MLCAS [14] provides an extension of XQuery to allow users to query an XML document without full knowledge of the structure. We experimentally compare our approach with the three approaches in [16][5][14] in Section 6. In [19] the concept of Smallest Lowest Common Ancestor (SLCA) is used to assign semantics to keyword queries. SLCAs are defined to be LCAs that do not contain other LCAs. This semantics is similar to that of the MLCAS approach.

In order to cope with low precision some approaches extend the database techniques with information retrieval techniques. In this direction, they rank the answers of keyword search queries on XML documents according to their estimated relevance [5][8]. Information retrieval systems using ranking functions may trade recall for precision. We view our keyword query language as database query language. Therefore, we do

not employ any ranking functions. Our goal is to not miss any meaningful answer and to exclude as many meaningless answers as possible.

Some languages employ approximation techniques to search for answers when the initial query is too restricted to return any. They either relax the structure of the queries or the matchings of the queries to the data [122]. In contrast to our language, these languages return approximate (not exact) answers.

Several papers focus on providing efficient algorithms for evaluating LCAs for keyword queries [1610514199]. Our approach is different and does not have to explicitly compute LCAs of nodes in the XML tree. In contrast, it computes a number of meaningful TPQs for keyword queries from a structural summary of the data tree (index graph). TPQs can be evaluated directly using an XQuery engine.

3 Data Model and Keyword Queries

We present in this section the data model and our simple keyword-based query language.

3.1 Data Model

Let \mathcal{E} be an infinite set of *elements* that includes a distinguished element r , and \mathcal{V} be an infinite set of *values*. Symbols e , and v (possibly with indices) refer systematically to an element, and a value respectively.

As is usual, we model XML documents as trees. Nodes in an XML tree are labeled by elements or values. In particular, the root node of an XML tree is the only node labeled by element r . Values can label only leaf nodes. For simplicity we assume that the same element does not label two nodes on the same path. Further, attributes of elements in an XML document are modeled as (sub)elements.

Figure 1 shows three XML trees T_1 , T_2 , and T_3 from different data sources that record bibliographic information in different formats (a slight extension of an example introduced in [14]). T_1 and T_3 categorize the data based on the publication year, while T_2 categorizes the data based on the type of publication (article or book). Still, in T_1 the year of the publications is specified as a child element of a ‘bib’ node, while in T_3 there is no ‘bib’ node, and the ‘book’ and ‘article’ nodes are children of a ‘year’ node that indicates their year of publication.

We are interested in retrieving information by issuing the same keyword query against all these data sources, even though information is structured differently in each one of them. Therefore, we view all these XML trees as one tree T rooted at r .

3.2 Keyword Query Language

Our query language allows the specification of element keywords and value keywords associated with elements (atomic predicates on elements).

Definition 1. A keyword query is a set of constructs each of them being an expression of the form: (a) an element e , or (b) a predicate $e = V$, where V , the annotation of e , is a set of values $\{v_1, \dots, v_k\}$, $k \geq 1$.

Suppose that we want to find the title and year of publications authored by “Mary” [14]. We formulate this request as the keyword query $Q_1 = [\text{year}, \text{author} = \{\text{Mary}\}, \text{title}]$. We use Q_1 as a running example on this paper.

The answer of a keyword query is based on the concept of query embedding.

Definition 2. An embedding of a keyword query Q to an XML tree T is a mapping M of the elements of Q to nodes in T such that: (a) An element e of Q is mapped by M to a node in T labeled by e , and (b) If an element e has an annotation V (that is, a predicate $e = V$ is specified in Q), then the image of e under M has a child value node labeled by a value in V .

We initially define keyword query answers as follows:

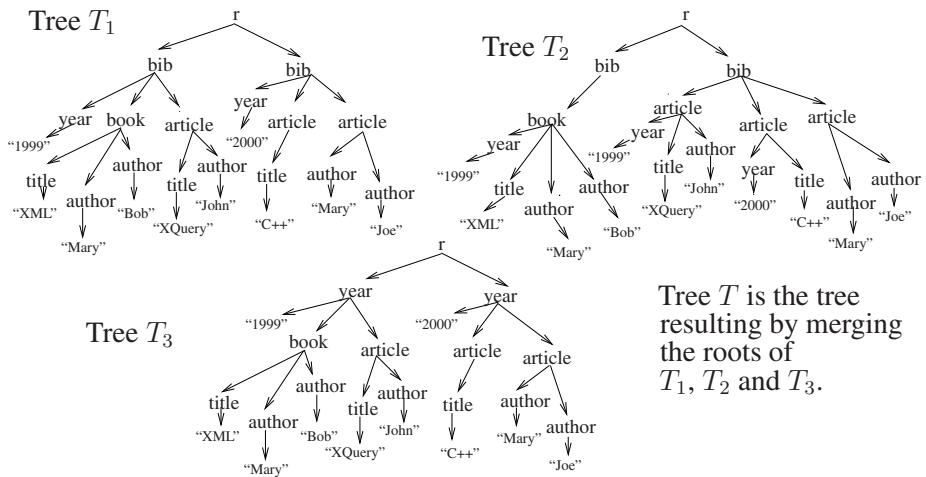
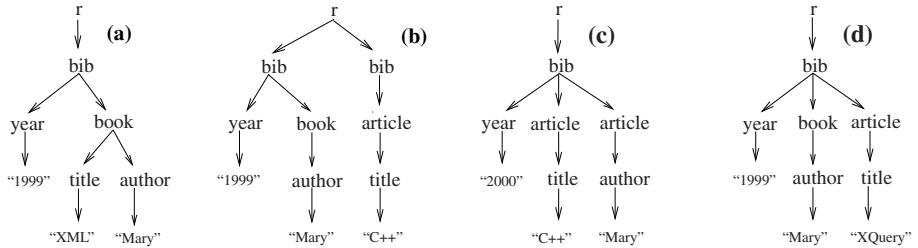


Fig. 1. An XML Tree T

Definition 3. Let Q be a keyword query and T be an XML tree. An answer of Q on T is a subtree T_a of T rooted at r such that: (a) there is an embedding M of Q to T_a , and (b) the leaf nodes of T_a are images under M of an element of Q or the child nodes of the image of an element node in Q (if it has one). The set of all the answers of Q on T is called answer set of Q on T .

Figure 2 shows four of the answers of the keyword query Q_1 on the XML tree T of Figure 1. More specifically, these answers correspond to embeddings of Q_1 to the XML tree T_1 . The keyword query is able to retrieve with one query the title and year of the publications of Mary from different parts of the XML tree, even though these parts structure the data in different ways.

The previous definition of the answer set of a keyword query accepts any possible embedding of Q to T . This generality allows embeddings that do not relate elements and values in the way the user was expecting when formulating the query. We call the answers corresponding to these embeddings *meaningless* answers. For instance, the

**Fig. 2.** Four answers of Q_1 on T

answer of Q_1 in Figure 2(a) correctly corresponds to a publication (a book in this case) authored by “Mary”. However, this is not the case with the answers of Q_1 shown in Figures 2(b), (c) and (d). In each one of them, year and/or title values do not correspond to a publication authored by “Mary” even though these values appear in an answer with “Mary”. In Section 5 we will present a technique that excludes these subtrees and returns answers to the user that are meaningful.

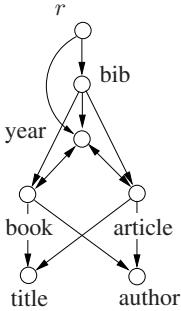
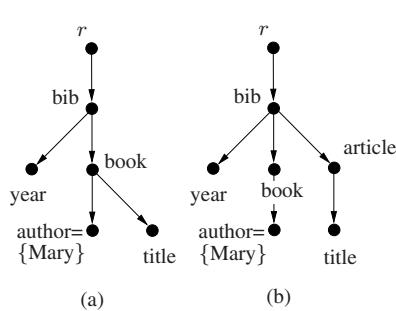
4 Evaluating Keyword Queries Using TPQs

We show now how keyword queries can be evaluated using TPQs. We first discuss index graphs for XML trees. Then, we use index graphs to construct a set of TPQs whose answers together form the answer set of a given keyword query. The TPQs of a keyword query provide the basis for defining meaningful semantics for such queries in the next section.

4.1 Index Graphs

Given a partitioning of the nodes of an XML tree T , an index graph for T is a graph G such that: (a) every node in G is associated with a distinct equivalence class of element nodes in T , and (b) there is an edge in G from the node associated with the equivalence class a to the node associated with the equivalence class b , iff there is an edge in T from a node in a to a node in b . Index graphs have been referred to with different names in the literature including “path summaries”, “path indexes” and “structural summaries”. They differ in the equivalence relations they employ to partition the nodes of the XML tree which includes simulation and bisimulation [15,13] or even semantic equivalence relations [17]. Index graphs have been extensively studied in recent years in both the “exact” [7,15,3] and the “approximate” flavor [13]. A common characteristic of these approaches is that the index graph is used as a back end for evaluating a class of path expressions without accessing the XML tree. To this end, the equivalence classes of the XML tree nodes are attached to the corresponding index graph nodes. Here we define index graphs where the equivalence classes are formed by all the nodes labeled by the same element in the XML tree. Figure 3 shows the index graph G of the XML tree T of Figure 1.

In contrast to other approaches, the equivalence classes of the XML tree nodes are not kept with the index graph. Therefore, keyword queries are ultimately evaluated on the

**Fig. 3.** Index graph G **Fig. 4.** Two TPQs of Q_1 on G : (a) U_1 , (b) U_3

XML tree. We use index graphs to support the evaluation of a keyword query through the generation of an equivalent set of TPQs.

4.2 TPQs for a Keyword Query

If G is the index graph of an XML tree T , we say that T *underlies* G . Given a keyword query Q and an index graph G , Q can be evaluated by computing a set of TPQs whose answers taken together are equal to the answer of Q on any XML tree underlying G . Intuitively, a TPQ satisfies both: the keyword query requirements and the structural constraints of the index graph.

Definition 4. Let Q be a keyword query and G be an index graph. A TPQ of Q on G is a TPQ U without descendant precedence relationships which is rooted at a node labeled by r and satisfies the following conditions:

- (a) There is a mapping M from the elements of Q to the nodes of U such that element e is mapped by M to a node labeled by e . If V_1, \dots, V_k are the annotations of all the elements in Q that are mapped to the same node n in U , n is annotated by $V_1 \cap \dots \cap V_k$. Two nodes in a path in U are not labeled by the same element, and every leaf node of U is the image of an element of Q under M .
- (b) There is a mapping M' from the nodes of U to the nodes of G that respects labeling elements and child precedence relationships. \square

Figure 4 shows two of the TPQs of the keyword query Q_1 on the index graph G of Figure 3.

We define an *answer* of a TPQ Q on an XML tree T to be a subtree of T which matches Q . The child value nodes of the matching element nodes of T are also included in an answer. The *answer set* of a TPQ on T is the set of all the answers of Q on T . The following proposition shows that the answers of a keyword query Q on an XML tree T can be computed by determining all the TPQs of Q on the index graph of T and by computing their answers on T .

Proposition 1. Let Q be a keyword query, G be an index graph, and U_1, \dots, U_k , $k \geq 1$, be all the TPQs of Q on G . Let also A, A_1, \dots, A_k be the answer sets of Q, U_1, \dots, U_k , respectively, on an XML tree underlying G . Then $A = \bigcup_{i \in [1, k]} A_i$. \square

Consider the XML tree T (Figure 1) and its index graph G (Figure 3). Consider also our keyword query Q_1 and two of its TPQs on G , U_1 and U_3 , shown in Figure 4. One can see that the answer of Q_1 on T shown in Figure 2(a) is also an answer of U_1 . Similarly, the answer of Q_1 on T shown in Figure 2(d) is also an answer of U_3 .

5 Using TPQs to Define Meaningful Answers

In this section, we assign semantics to our keyword query language that returns meaningful answers. In contrast to previous approaches which exclude embeddings of the query to the XML tree [16, 5, 14, 19], our approach excludes TPQs of the query on the index graph of the XML tree. In this sense, our approach relies both on data and on structural patterns of data, instead of relying exclusively on data.

Based on the results of the previous section, we consider that, given an XML tree T (and its index graph G), the answer of a keyword query on T is the union of the answers of its TPQs on G . However, some of these TPQs may return meaningless answers. Consider, for instance, our query Q_1 and the XML tree of Figure 1 along with its index graph G of Figure 3. The TPQ U_3 of Q_1 on G shown in Figure 4(b) returns (among others) the answer shown in Figure 2(d). This answer is meaningless. Therefore, TPQ U_3 should not be used for computing the answers of Q_1 on T . Analogously to query answers, we characterize a TPQ of a query Q on G as *meaningful* with respect to T if it returns a *meaningful* answer on T . Otherwise, it is characterized as meaningless with respect to T . In order to formally define meaningful TPQs we need to introduce a transformation for TPQs.

5.1 A Transformation for TPQs

Let Q be a keyword query, T be an XML tree and G be its index graph. Figure 5 shows two TPQs, U and U' , of Q on G . TPQ U comprises three subtrees T_a , T_b and T_c rooted at the nodes labeled by a , b , and c respectively. T_b is the subtree of T_c . Subtrees T_a and T_b can be empty (that is, trivially contain only their root node a and b respectively). The c -node can coincide with the root of U . However, the a -node cannot coincide with the c -node, and the b -node cannot coincide with the c -node (that is, the c -node is an

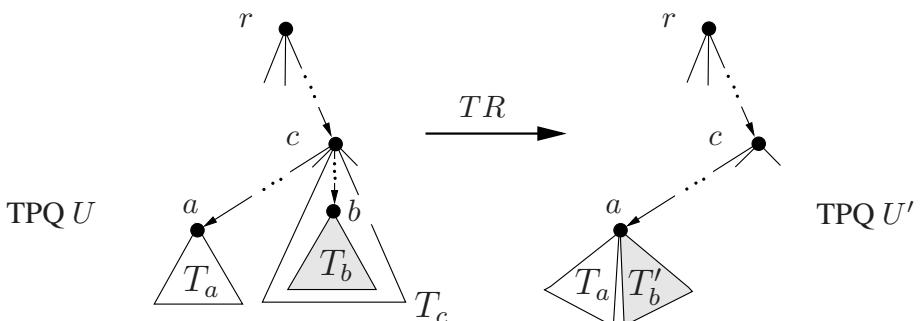


Fig. 5. Transformation TR

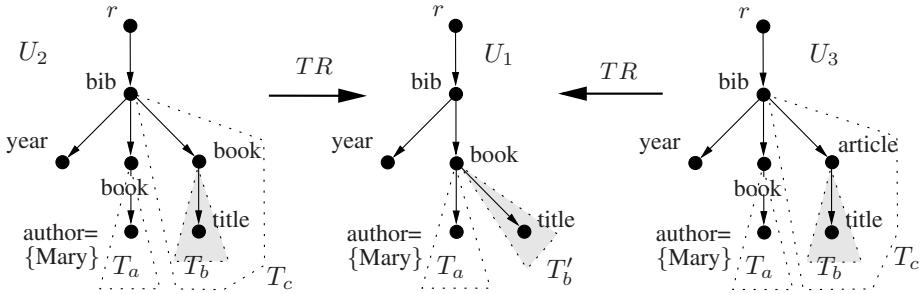


Fig. 6. Two applications of transformation TR to TPQs of Q_1 on G . U_2 and U_3 are meaningless.

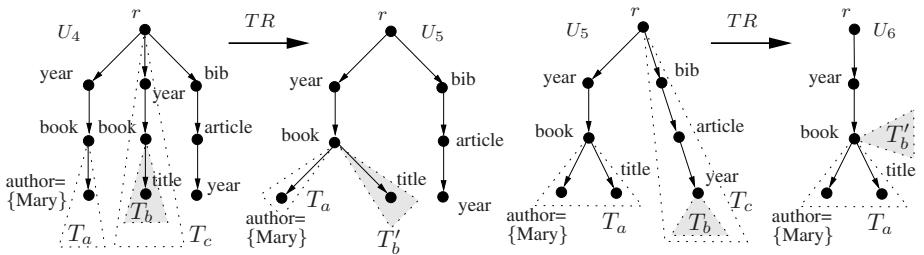


Fig. 7. Two applications of TR to TPQs of Q_1 on G , in sequence. U_4 and U_5 are meaningless.

ancestor of the a -node and b -node). Labels a and b can be equal. Subtree T'_b in U' is a tree identical to T_b except that its root is labeled by a instead of b . TPQ U' can be obtained from U by removing the subtree T_c below the c -node, and by making T'_b a subtree of the a -node. The transformation TR on TPQs is a transformation that takes a TPQ of the form of U and returns a TPQ of the form of U' .

Consider, for instance, the keyword query Q_1 and the index graph G (Figure 3). Figure 6 shows three TPQs U_1 , U_2 and U_3 of Q_1 on G , and two applications of transformation TR . Dotted lines denote the subtrees T_a , T_b , and T_c of transformation TR as they are graphically shown in Figure 5. Notice that in U_2 , the roots of T_a and T_b are labeled by the same element ‘book’, while in U_3 they are labeled by different elements ‘book’ and ‘article’. Figure 7 shows two applications of TR in sequence. Notice that T_b of U_5 (and consequently T'_b of U_6) are empty. TPQ U_5 has also an extra branch from the root with respect to U_6 .

5.2 Determining Meaningful TPQs

We first provide some intuition on transformation TR . Consider a TPQ U' resulting by applying TR to a TPQ U . To understand the idea, observe that there is a 1-1 mapping f from the nodes of U' to the nodes of U that respects node labels and child precedence relationships (with the exception of the child precedence relationships from the node labeled by a in T'_b). The following proposition holds:

Proposition 2. Assume that TPQ U' results by applying transformation TR to a TPQ U . If n' is the lowest common ancestor (LCA) of the nodes n'_1, \dots, n'_k in U' , and n is the LCA of the nodes $f(n'_1), \dots, f(n'_k)$ in U then n is not a descendant of $f(n')$ in U . \square

Since, there is an answer of Q on T that closely relates the nodes as determined by U' , any answer of Q on T that relates the nodes in the looser way determined by U is not meaningful. Therefore, if U' returns an answer on T , U should be characterized as meaningless and should be excluded from generating an answer for Q on T .

Definition 5. A TPQ U of Q on G is called *meaningless with respect to T* if there is another TPQ U' of Q on G such that (a) U' can be obtained from U by a sequence of applications of transformation TR , and (b) U' has an answer on T . Otherwise, it is called *meaningful with respect to T* . \square

Consider the TPQs U_1, U_2 and U_3 of Q in G shown in Figure 6. One can see that U_1 has an answer on T . Therefore, U_2 and U_3 are meaningless w.r.t. T . Consider also the TPQs U_4, U_5 and U_6 of Q on G shown in Figure 7. One can see that U_5 and U_6 have an answer on T . Therefore, U_4 and subsequently U_5 are meaningless w.r.t. T .

We can now update the definition of the answer set of a keyword query given in Section 3.2 so that an answer set comprises only meaningful answers. The new definition is based on Proposition 1 and Definition 5.

Definition 6. Let Q be a keyword query, T be an XML tree and G be an its index graph. Let also U_1, \dots, U_k , $k \geq 1$, be the meaningful TPQs of Q on G with respect to T . If A, A_1, \dots, A_k are the answer sets of Q, U_1, \dots, U_k , respectively, on T , then $A = \cup_{i \in [1, k]} A_i$. \square

Consider the TPQ U_3 of Q_1 on G shown in Figure 4(b). As mentioned in Section 4.2, U_3 evaluated on the XML tree T of Figure 1 returns the meaningless answer of Figure 2(d). TPQ U_3 is also shown in Figure 6 and it is meaningless according to Definition 5. Therefore, it will not be used to generate answers for query Q_1 on T . In contrast, TPQ U_1 of Figure 4(a) returns only the meaningful answer of Figure 2(a). TPQ U_1 is also shown in Figure 6. One can see that TR cannot be applied to U_1 . Therefore, it is correctly characterized by Definition 5 as meaningful, and will be used to generate answers for Q_1 on T .

Since the meaningful TPQs are TPQs, their evaluation can be implemented on top of an XQuery engine and benefit from the extensive optimization techniques that have been developed up to now for XQuery [11][4].

6 Experimental Evaluation

We implemented our approach (*Meaningful Tree Pattern - MTP*) and the three other approaches Meet [16], XSEarch [5], and MLCAS [14]. We ran detailed experiments to compare their *Recall* (defined as the proportion of relevant materials retrieved) and *Precision* (defined as the proportion of retrieved materials that are relevant).

We used real-world DBLP data collected in May 2006. To reduce the size of the document for the experiments, we retained only three publication types: ‘book’, ‘article’,

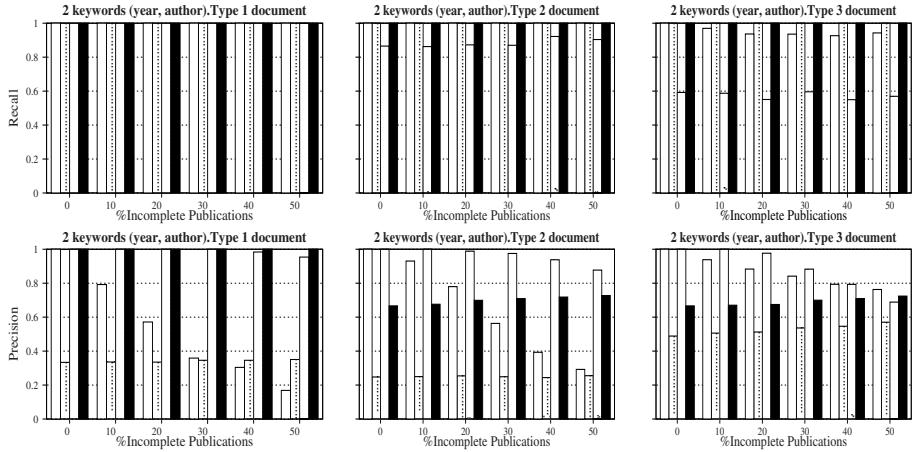


Fig. 8. Recall and Precision for the two-keyword query {author, year}

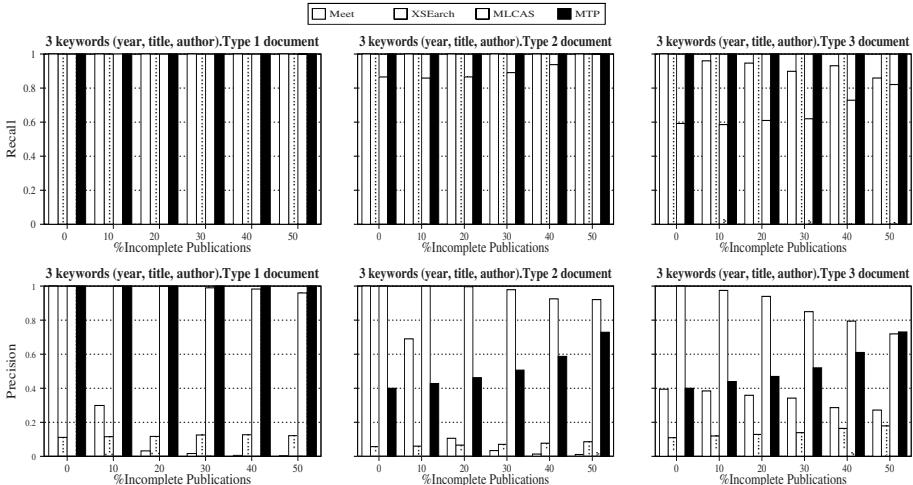


Fig. 9. Recall and Precision for the three-keyword query {title, author, year}

and ‘inproceedings’. For each publication type, we retained only the properties ‘title’, ‘authors’, and ‘year’. As the original DBLP data is flat, we restructured it into three types of data sets. Publications in schema type 1 do not have references. Publications in schemas type 2 and 3 may have references. One difference between schemas type 2 and type 3 is that publications in schema type 3 are categorized by year.

Besides the structure of the document, the “incompleteness” of the data also affects the effectiveness of the keyword based searches. We define a publication in the data set as *complete* if it has all the subelements ‘title’, ‘year’, and ‘author’. Otherwise it is *incomplete*. For each query and each data set type, we ran the four approaches on six XML

documents with increasing percentage of incomplete publications in the range from 0% (all the publications are complete) to 50% (half the publications are incomplete).

We ran the experiments on a Pentium 2.40GHz computer with 512MB of RAM running Windows XP Professional. We implemented all keyword search techniques in Java and used the SAX API of the Xerces Java Parser for the parsing of XML files. Berkeley DB XML 2.2.13 was used to store XML files and run XQuery.

Figure 8 shows precision and recall of the two-keyword query {author, year} for the three types of documents varying the percentage of incomplete publications in the documents. Figure 9 reports on the same measurements for the three-keyword query {title, author, year}. The trends are similar with a slight degradation of the recall of the Meet approach, and an average degradation of the precision of XSEarch and Meet.

In summary, Meet and XSEarch show very poor precision on the average. MLCAS improves significantly on precision but scores low on recall both for the two- and the three-keyword query. This performance is not satisfactory for a database query language. Employing a structured TPQ language (e.g. XQuery) to further filter a query answer set using structural restrictions does not recover the missed meaningful answers. In contrast, MTP shows perfect recall. It also shows better precision compared to approaches with similar recall. Precision can be further improved by imposing structural restrictions on the answer set or by integrating our semantics for keyword queries with a structured TPQ query language.

7 Conclusion

Issues related to applications exporting and exchanging XML data on the web have motivated recently the extension of keyword-based techniques for querying XML documents. Although these keyword-based approaches provide independence from the structure of the XML documents, they fail to retrieve all and only meaningful answers especially when the XML data are incomplete.

We have introduced a simple keyword query language for querying XML documents and we suggested a novel semantics for it. In contrast to previous approaches that operate locally on data to extract lowest common ancestors (LCAs), our approach operates on structural summaries of data to extract meaningful tree pattern. This global view of data provides an advantage to our approach compared with previous ones. Our approach generates tree pattern queries TPQs. Therefore, it can be easily implemented on top of an XQuery engine and benefit from well known query optimization techniques. We experimentally compared our approach to previous ones. Our experimental evaluation shows that it has a perfect recall both for XML documents with complete and incomplete data. It also has a better precision compared to approaches with similar recall. Its precision can be further improved by further specifying structural restrictions on the answers returned by a keyword query.

We are currently working on applying the semantics suggested in this paper for keyword search to recently suggested query languages for tree structured data [17][18] that flexibly allow not only keywords but also partial specification of a tree pattern.

References

1. S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *Proc. of the Intl. Conf. on Data Engineering*, pages 141–, 2002.
2. S. Amer-Yahia, S. Cho, and D. Srivastava. Tree Pattern Relaxation. In *Proc. of the 8th Intl. Conf. on Extending Database Technology, Prague, Czech Republic*, 2002.
3. A. Barta, M. P. Consens, and A. O. Mendelzon. Benefits of Path Summaries in an XML Query Optimizer Supporting Multiple Access Methods. In *Proc. of the 31st Intl. Conf. on Very Large Data Bases*, pages 133–144, 2005.
4. N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 310–321, 2002.
5. S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSearch: A Semantic Search Engine for XML. In *Proc. of the 29th Intl. Conf. on Very Large Data Bases*, 2003.
6. D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into xml query processing. *Computer Networks*, 33(1-6):119–135, 2000.
7. R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proc. of the 23rd Intl. Conf. on Very large Databases*, pages 436–445, 1997.
8. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 16–27, 2003.
9. V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword Proximity Search in XML Trees. *IEEE Trans. Knowl. Data Eng.*, 18(4):525–539, 2006.
10. V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword Proximity Search on XML Graphs. In *Proc. of the 19th Intl. Conf. on Data Engineering*, pages 367–378, 2003.
11. H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. Timber: A native XML database. *VLDB Journal*, 11(4):274–291, 2002.
12. Y. Kanza and Y. Sagiv. Flexible Queries Over Semistructured Data. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2001.
13. R. Kaushik, P. Bohannon, J. F. Naughton, and H. F. Korth. Covering Indexes for Branching Path Queries. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 2002.
14. Y. Li, C. Yu, and H. V. Jagadish. Schema-Free Xquery. In *Proc. of the 30th Intl. Conf. on Very Large Data Bases*, pages 72–83, 2004.
15. T. Milo and D. Suciu. Index structures for Path Expressions. In *Proc. of the 9th Intl. Conf. on Database Theory*, pages 277–295, 1999.
16. A. Schmidt, M. L. Kersten, and M. Windhouwer. Querying XML Documents Made Easy: Nearest Concept Queries. In *Proc. of the 17th Intl. Conf. on Data Engineering*, 2001.
17. D. Theodoratos, T. Dalamagas, A. Koufopoulos, and N. Gehani. Semantic Querying of Tree-Structured Data Sources Using Partially Specified Tree-Patterns. In *Proc. of the 14th ACM Intl. Conf. on Information and Knowledge Management*, pages 712–719, 2005.
18. D. Theodoratos, S. Soulardatos, T. Dalamagas, P. Placek, and T. Sellis. Heuristic Containment Check of Partial Tree-Pattern Queries in the Presence of Index Graphs. In *Proc. of the 15th ACM Intl. Conf. on Information and Knowledge Management*, 2006.
19. Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 2005.

Lightweight Model Bases and Table-Driven Modeling

Hung-chih Yang¹ and D. Stott Parker²

¹ Yahoo!, Sunnyvale, CA 94089, USA

hcyang@cs.ucla.edu

² UCLA, Los Angeles CA 90024, USA

stott@cs.ucla.edu

Abstract. Data mining models are often implemented as program code and stored in an ad hoc fashion. In this paper we describe a methodology for developing model bases that can be implemented with only an extended relational database. This method stores models as what we call *lightweight functions*, which are straightforward textual representations of function values.

Many applications use models that admit this approach. Business applications in particular, which can have a very large number of special-case rules or business logic, are suitable for development with lightweight functions. Financial and forecasting applications give another example. We argue that the Lightweight Model Base offers some advantages for these applications.

Introduction of lightweight stored functions in relational databases is a way to integrate the software-engineering methodology of *table-driven programming*. This methodology advocates storing functions and data in tables. The computing process is just a mechanical evaluation of “joining” data and function relations. It would make stored business logic transparent for understanding and maintenance as relational data. Table-driven programming has much in common with statistics and data mining, and is a natural framework for combining data mining with databases.

1 Introduction

One of the authors had once participated in a decision-support project that involved with thousands of forecasting models. These models were straightforward mathematical formulas generated by a statistical software package and stored in a relational database. They were routinely joined with a collection of data relations, then evaluated to produce important business forecasts. The formulas were originally created in text files, and first a parser was used to separate coefficients, variables, and function calls from the formula structures. Later the separated formula components were stored in several normalized relations in the forms of numbers, strings, and IDs. Formula structures were stored in a relation as BLOBs. An external forecasting process routinely joined several data relations

with the formula relations and rebuilt forecasting formulas with plugged-in variable values. Forecast values were then computed in the external process and stored back in a relation. In order to reduce network communication, function relations and data relations were queried separately and a C++-implemented forecasting process had to simulate an in-memory sort-merge **join** in order to put data and reconstructed functions together. This experience has made us wonder: why can't a relational database stores the formulas in a relation directly and have the formula tuples joined with data tuples inside the database server to generate results?

1.1 Lightweight Models

The formulas below are models for forecasting future demand. They were generated by a time-series package in the aforementioned decision-support project. In these models, *DMD_FCST_F* represents the demand that we would like to get a forecast on. At the right-hand-side of these formulas, *var_rh*, *var_rh(KEY('11'))*, *var_ih*, *var_vilav*, and *var_high*, etc. are some current and past time-series variables.

$$\begin{aligned}
 DMD_FCST_F = & -2.9131 + (0.8268 * var_rh) + (2.4243 * var_ih) + \\
 & (-0.5168 * (var_rh - var_rh(KEY('11')))) + (0.0437 * var_vilav) + \\
 & (-17.1098 * var_cl) + (6.4059 * var_high) \\
 DMD_FCST_F = & 6.3184 + (2.8979 * var_rh) + (0.0362 * var_vilav) \\
 & + (-16.7509 * var_cl) + (-12.2987 * var_low) \\
 DMD_FCST_F = & -4.5242 + (2.4416 * var_rh) + (0.8179 * (var_rh - var_rh(KEY('9')))) + \\
 & (0.0447 * var_vilav) + (8.9951 * var_high)
 \end{aligned}$$

Fig. 1. Examples of forecast formulas

In a broader perspective, models usually consist of rules, conditions, associations, processes, formulas, constraints, and regulations. Modern databases can easily store models like these formulas, but as a data management problem, modeling is *heavy-duty*, in that it is usually handled in a relational database by a conjoined form of external processes, stored procedures, user-defined functions, middleware components, object types and methods, constraints, and triggers, etc. Thus, stored models are generally unique in structure, dependent on one another, and complex but limited in scope.

On the other hand, modern decision-support and data mining systems can automatically generate models, association rules, and classification rules in huge volume. This resulting model instances usually have similar but not identical structure. Thus, we cannot simply use one function to represent them all, and rely on input arguments for diversity. These models are *lightweight*, in that it is simpler in nature, and using the aforementioned database objects to handle them would be overkill: because of the huge volume and lightweight nature of business logic, applying heavy-duty object types or stored functions to represent them can create overhead and make it difficult to access these database objects

in a relational manner. Thus, in this paper we will discuss the problem of how to properly store and utilize many lightweight models in a relational way.

1.2 The Table-Driven Modeling Concept

In fact, the functionality we expected to see from a RDBMS is *table-driven programming* [4]. It is a software methodology popular in script programming and in tools such as parsers. Its basic principles are (a) data and instructions are stored and managed indistinguishably (except perhaps for their types) in tables, and (b) programming logic is controlled by using conditions (or states) that select data and instructions from the tables for execution. The mechanism of selecting data and instructions is generally a straightforward mechanical process. This process can be implemented in a declarative relational query and realized in a RDBMS execution plan. Individual aspects of programming logic can be implemented with relational select, join, filter, or set operations, etc. A limited form of using data only to direct the program logic can be summarized by the *data-driven* [8] methodology.

Even though modern databases can already store programming logic as stored procedures or object methods, these database objects are undeniably ‘heavy’ for dealing with simple rules, particularly when there are a large number of them. Stored procedures and methods are created to be part of system catalog, and are managed with imperative DDL statements that require special privileges. In our opinion, straightforward data mining rules ought to be managed like data — using DML (user-oriented data manipulation language) instead of DDL (administrator-oriented data definition language).

A similar concern was also expressed in the Lowell Report [3] that “code is not data.” In modern databases, there are already many constructs that store programming logic: stored functions, triggers, user-defined functions, and object types, etc. However, they have all been introduced as a new database object — in the same class as tables that is part of a schema, instead of as data in tuples. This database-object-centric design reserves them for database administrators and developers and limits managing them from database users. In ordinary situations, database users cannot create new database objects (e.g., object types) on their own. Even though database designers can still implement these database objects for storing data-mining rules, they hardly can be relationally integrated with data.

In a *pure relational* environment, data as well as functions are associated relationally, i.e., functions are data. We call this kind of function-value-as-data *lightweight function*. Lightweight functions can be a primitive datatype. Function values can be inserted into a table using DML statements, and can be easily replaced with an update statement. This approach differs significantly from creating a new object type where method definitions are created as part of the type creation DDL, and cannot be easier modified using DML. ORDBMS stored functions and object methods may be stored as data in system catalog, but from a database user’s perspective, they are no difference from built-in functions. On the other hand, using the table-driven methodology and lightweight functions,

data mining rules or business logic can be managed and evaluated within a relational database server.

1.3 Lightweight Model Bases

Data mining models are usually maintained within data-mining applications, but we believe it can be better managed in a database management system along with the data for mining. Some reasons for this include:

- Data mining applications can create many rules in the form of simple expressions. Managing large numbers of these rules can become a problem for data mining applications, while databases provide a relatively unlimited repository for storing, managing, and evaluating these rules.
- Rules stored in a proprietary format can be non-portable among data-mining systems, while using database utilities, it is relatively easy to transform and transport rules stored in databases.
- Inspecting, managing, and querying data-mining rules can be difficult from one proprietary to another data mining environment, while relational databases provide a standard, uniform, and straightforward data model and query language (e.g., SQL).
- If data mining rules are extensively used in an enterprise system, then databases can extend all the features already built-in for data management to metadata and model management. These features include transaction management, backup/restore services, concurrent accesses, scalable storage, and indexing, etc.

In essence, we advocate using relational databases as *model bases* that can store, manage, and evaluate lightweight data mining models.

The rest of this paper is organized as follows. Section 2 gives a detailed introduction to the algorithms and techniques needed for table-driven modeling. A sophisticated example is then implemented with recursive SQL in section 3, showing how to implement a loan application system using decision trees. Section 4 describes how to emulate lightweight functions using heavy-duty ORDBMS objects. Conclusions are in section 5.

2 Table-Driven Modeling

Table-driven is a methodology that advocates storing pieces of programming logic in tables and querying them for execution based on selection conditions. This section describes using RDBMS relations as the “tables” to store a large amount of auto-generated business rules and models implemented in lightweight functions. A SQL query can then join model and data relations and run these models to produce results.

2.1 Table-Driven Programming in SQL

One great benefit brought by relational databases is the clear representation of data and their relationships, but this benefit does not extend to traditional

stored procedures and functions. Unlike relational tables, the imperative nature of stored procedures is navigational. Convoluted subroutine calls make a program hard to understand and maintain, and program logic can be hidden beyond recognition. Object-Oriented (or Object-Relational) programming can introduce meandering navigational behavior into communications among objects. In order to make program logic more transparent, we advocate the introduction of table-driven programming in relational databases. The goal is to transform the communications among functional modules from explicit navigation in stored procedures to joins of small pieces of programming logic in a declarative relational manner.

The design process of a table-driven system contains two stages: *factor* and *assemble* (or *join*). Since this two-stage process is to apply the relational model on programming code, we call it *relational programming* as well.

- **Factor** — This stage is to split a monolithic system of business logic or data-mining knowledge into simple rules and formulas that can be stored in attributes of normalized relations.
- **Assemble (or Join)** — In the second stage, users can write declarative queries to join tables of rules, formulas, and data together to find results.

Notice that in both stages, we utilize only SQL constructs: functions of SQL expressions in the factor stage for storing granular program logic, and SQL queries in the assemble stage for running a table-driven computing process. An assemble query have traits of meta and functional programming.

A developer may start a software project following a traditional database design and analysis process, such as the ER-diagram [1]. In a schema, some of the relations may contain function attributes. Using this schema, program logic is built within queries and views that join data and functions together, rather than coding complicated imperative stored or external procedures. Individual rules and formulas can be easily replaced and redefined dynamically, while the overall picture of the program logic stays within easy-to-understand queries or views. Individual pieces of program logic can be separately maintained independent from the queries. In a traditional database design, the whole program logic and subroutines may need rewriting if there is any specification change (even a minor one), and only data can be dynamically changed (because data is defined relationally, but not code). The ultimate goal of relational programming is to transform programming from an obscure, imperative, and arbitrary form to a transparent process of relational operations. This might not be as powerful and versatile as the traditional programming process, but when we are dealing with a vast amount of simple program logic, it's a process worth considering.

Data models and database paradigms have often flirted with religion in the exploration of methodologies. The approach proposed in this paper is very practical, and even ecumenical in scope (in the sense that it can be applied within any religion). It is also theoretically straightforward, so its exposition does not really benefit from formalism. So, in this paper we have decided to ground the concepts in examples, so that their practicality and usefulness is as apparent as possible, to as wide an audience as possible.

2.2 Lightweight Functions

Table-driven programming can be implemented using existing ORDBMS polymorphic object methods. Pieces of programming code are embedded in objects and they can be selected and executed dynamically in queries. However this approach have some shortcomings:

- **Object types are quite heavy.** Object types are created and maintained in data definition language (DDL) and stored in system catalog. A table-driven application may possess thousands or even millions of items of program logic, therefore creating millions of object types to wrap these pieces of logic can cause great burden on the database system catalog.
- **Code wrapped in objects is still not data.** Because code wrapped in methods are attached to objects, they are not data [3] in the sense that they cannot be accessed or manipulated like ordinary data.
- **Dynamic dispatch is complex.** From the manuals of ORACLE [6] and DB2 [5], looking for the right method implementation in a type hierarchy can be done in two fashions: top-down or bottom-up respectively for ORACLE and DB2. Either way is quite complex and could be time consuming (see section 4.1).

Based on these issues, we believe RDBMS needs an alternative called *lightweight functions* in order to support table-driven modeling efficiently. Briefly, this idea is to store *function expressions* as values in attributes of a relation and extend the type system of a relational database to include function datatypes — aggregation of function arguments and a return type. These function attributes can then be joined and applied with data attributes in a query. During a join operation, function attributes are evaluated like any expressions allowed in the select-list (or where-clause) of an SQL query. There are two designs for defining a function datatype as a table attribute:

- **weak-typed**

In a weak-typed design, a *FUNCTION* type is added to the RDBMS primitive datatype collection. The signature of a function attribute is not overt to the database system. Only at the run time (in a query), the database execution engine can match the input arguments and parameters of stored lightweight functions together for evaluation.

- **strong-typed**

For a strong-typed design, a function type is expressed as a collection of parameter datatypes and a return datatype. A SQL parser can do type checking during inserting tuples with function attributes. Below is an example of a strong-typed function datatype:

$$\text{VARCHAR} \times \text{NUMBER} \times \text{VARCHAR} \rightarrow \text{CHAR}(3)$$

The weak-typed design is flexible for accommodating a diversity of lightweight functions whose signatures may not be known during schema design. The trade-offs for both designs have been discussed thoroughly in programming language circle and sometimes it becomes almost a religious issue and we won't go any

further in this paper. In the rest of this paper we use the strong-typed design in examples and figures.

Definition of a function does not need to be small, but ought to constitute atomic functionality. There are several candidates that we can use to specify the syntax and functionality for the lightweight functions. These candidates are from the ample programming constructs already incorporated in a modern RDBMS. They include a) SQL expressions, b) stored functions, and c) user-defined functions. They have varying degrees of programming capability and expressiveness. For most lightweight program logic, the SQL expressions would suffice.

3 Example: A Loan-Application System

In this section, we present a sophisticated example of using table-driven stored functions in a recursive query. Deciding whether to approve a loan application can be implemented using a decision tree classifier [7]. Fig. 2 shows such an example decision tree. Given the background information of a loan applicant, the system starts from the root of a selected decision tree and follows a path of boolean expressions toward a final decision. A decision tree is obviously an important set of business logic for a financial institution. Even though most data of the loan-application system are probably stored in a relational database, the model of the decision tree itself is usually a monolithic process implemented internally or externally. This process may contain an aggregation of simple decision expressions, thus it's an ideal candidate for the software process of table-driven programming. In doing so, we can store decision tree branches in an attribute of tuples. These tuples are then linked with each others forming a hierarchical structure of a decision tree. A loan-application process is just a recursive join of the decision-tree tables and applicant information tables.

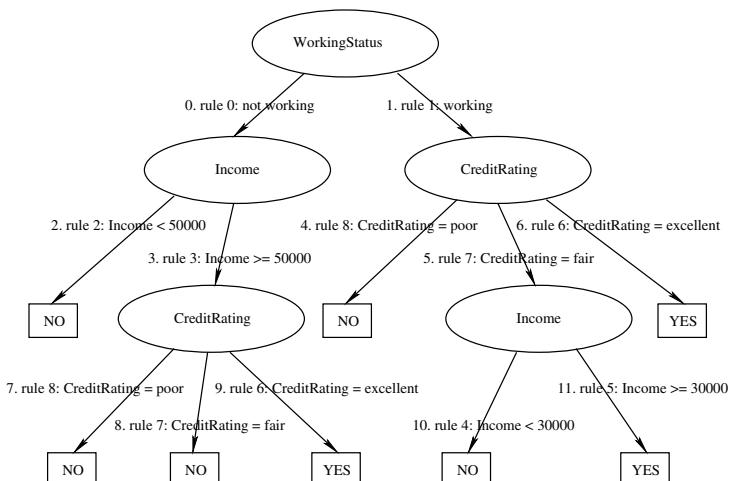


Fig. 2. An example of loan-application decision tree used in section 3.1

3.1 Schema Design

Three tables are defined for this system. The first table, Customer, stores loan applicants' background information including working status, income, and credit rating. This information would be used as arguments by the branch boolean expressions of a decision tree. The second table, ClassificationRule, has a function attribute Rule with the datatype of $(\text{VARCHAR}, \text{NUMBER}, \text{VARCHAR}) \rightarrow \text{CHAR}$. It is a function attribute that its instances take three parameters of two VARCHARs and one NUMBER. The actual arguments will be assigned during a query. In this demonstration the arguments are the three background attributes from the Customer table. The third table, DecisionTree, is a relationship table between the Customer and ClassificationRule tables. The DecisionTree table also stores the hierarchical structure of decision trees. SQL's recursive query [2] is just the tool to traverse a decision tree stored in DecisionTree. Sample data of these tables are listed respectively in Table 1 for Customer, 2 for ClassificationRule, and 3 for DecisionTree.

Fig. 3 is the DDL for creating the Customer table. Name is the primary key and the other attributes store the background information of loan-applying customers.

```
CREATE TABLE Customer(
    Name          VARCHAR(16) NOT NULL,  WorkingStatus VARCHAR(32) NOT NULL,
    Income        NUMBER        NOT NULL,  CreditRating  VARCHAR(32) NOT NULL,
    CONSTRAINT Customer_PK PRIMARY KEY (Name));
```

Fig. 3. DDL for creating the Customer table

To create the ClassificationRule table, a plausible DDL is listed in Fig. 4. The second attribute, Rule, stores the logic of individual decision branch. Rule is implemented as a table-driven lightweight function with three parameters. Even though the actual arguments are determined during a query, we stipulate that they are the three attributes of background information provided by the Customer table.

```
CREATE TABLE ClassificationRule(
    RuleID INTEGER           NOT NULL,
    Rule  (\text{VARCHAR},\text{NUMBER},\text{VARCHAR}) -> CHAR NOT NULL,
    Description VARCHAR(48),
    CONSTRAINT ClassificationRule_PK PRIMARY KEY (RuleID));
```

Fig. 4. DDL for creating the ClassificationRule table

The DDL for creating the DecisionTree is showing in Fig. 5. In this table, ParentBranchID is the attribute that defines a self-referencing relationship for tuples. The root-node branches of decision trees are the ones who do not have a parent (ParentBranchID is NULL). Each branch of a decision tree has a primary key, BranchID. The attribute, Decision, is used for final classification when there

```

CREATE TABLE DecisionTree(
    BranchID      INTEGER NOT NULL,    ParentBranchID  INTEGER,
    RuleID        INTEGER NOT NULL,    Decision        VARCHAR(4),
    Description    VARCHAR(128),
    CONSTRAINT DecisionTree_PK PRIMARY KEY (BranchID),
    CONSTRAINT ParentBranchID_FK FOREIGN KEY (ParentBranchID) REFERENCES DecisionTree(BranchID),
    CONSTRAINT RuleID_FK          FOREIGN KEY (RuleID) REFERENCES ClassificationRule(RuleID));

```

Fig. 5. DDL for creating the DecisionTree table

is no subtree to go further. In our example, the class designation is either “yes” or “no” for granting or denying a loan application. In our simplified example, we assume that all customers can be found a way down to a leaf node in the decision tree, thus internal tree nodes do not need a class designation. Also the RuleID attribute references ClassificationRule’s primary key. Using it, the actual boolean expression can be retrieved and used as a branch in a decision tree.

3.2 Querying and Decision Making

As indicated, a recursive query (see Fig. 6) can be used to traverse the DecisionTree table in order to determine whether to accept or deny a loan application. In this query, the customer information was first fetched in an initial query and passed down the hierarchy of a decision tree during each recursive join. In each join, the ClassificationRule.Rule function is applied on customer attributes to determine the next branch of a traversal decision path. Since no lightweight function type is implemented in any real-world RDBMS, the ClassificationRule table can be emulated using ORDBMS object types and its Rule attribute is emulated by a polymorphic object method. Based on this ORDBMS emulation and the sample data listed in Table 1, 2, and 3, the result of the recursive query is shown in Table 4.

```

WITH RECURSIVE
AncestorDT(BranchID,Decision,Name,WorkingStatus,Income,CreditRating) AS
((SELECT DecisionTree.BranchID,DecisionTree.Decision,Customer.Name,
Customer.WorkingStatus,Customer.Income,Customer.CreditRating
FROM Customer,ClassificationRule,DecisionTree
WHERE DecisionTree.ParentBranchID IS NULL
    AND DecisionTree.RuleID = ClassificationRule.RuleID
    AND ClassificationRule.Rule(Customer.WorkingStatus,Customer.Income,
Customer.CreditRating)=’T’)
UNION ALL
(SELECT DecisionTree.BranchID,DecisionTree.Decision,AncestorDT.Name,
AncestorDT.WorkingStatus,AncestorDT.Income,AncestorDT.CreditRating
FROM ClassificationRule,DecisionTree,AncestorDT
WHERE AncestorDT.BranchID = DecisionTree.ParentBranchID
    AND DecisionTree.SubRuleID = ClassificationRule.RuleID
    AND ClassificationRule.Rule(AncestorDT.WorkingStatus,AncestorDT.Income,
AncestorDT.CreditRating)=’T’))
SELECT DISTINCT Name,Decision
FROM AncestorDT
WHERE Decision IS NOT NULL;

```

Fig. 6. A recursive query for deciding loan applications

Table 1. Sample data for the Customer table

Name	WorkingStatus	Income	CreditRating
John	working	35000	fair
Tom	not working	60000	fair

Table 2. Sample data for the ClassificationRule table

Rule ID	Rule ((VARCHAR,NUMBER,VARCHAR) → CHAR)	Description
0	(WorkingStatus,Income,CreditRating) → (CASE WHEN WorkingStatus = 'not working' THEN 'T' ELSE 'F' END)	Return 'T' if the customer does not have a job.
1	(WorkingStatus,Income,CreditRating) → (CASE WHEN WorkingStatus = 'working' THEN 'T' ELSE 'F' END)	Return 'T' if the customer has a job.
2	(WorkingStatus,Income,CreditRating) → (CASE WHEN Income < 50000 THEN 'T' ELSE 'F' END)	Return 'T' if the customer's income is < 50000.
3	(WorkingStatus,Income,CreditRating) → (CASE WHEN Income ≥ 50000 THEN 'T' ELSE 'F' END)	Return 'T' if the customer's income is ≥ 50000.
4	(WorkingStatus,Income,CreditRating) → (CASE WHEN Income < 30000 THEN 'T' ELSE 'F' END)	Return 'T' if the customer's income is < 30000.
5	(WorkingStatus,Income,CreditRating) → (CASE WHEN Income ≥ 30000 THEN 'T' ELSE 'F' END)	Return 'T' if the customer's income is ≥ 30000.
6	(WorkingStatus,Income,CreditRating) → (CASE WHEN CreditRating = 'excellent' THEN 'T' ELSE 'F' END)	Return 'T' if the customer's credit rating is excellent.
7	(WorkingStatus,Income,CreditRating) → (CASE WHEN CreditRating = 'fair' THEN 'T' ELSE 'F' END)	Return 'T' if the customer's credit rating is fair.
8	(WorkingStatus,Income,CreditRating) → (CASE WHEN CreditRating = 'poor' THEN 'T' ELSE 'F' END)	Return 'T' if the customer's credit rating is poor.

Table 3. Sample data for the DecisionTree table

Branch ID	Parent Branch ID	Rule ID	Decision	Description
0	NULL	0	NULL	(not working) ⇒ Need a further decision.
1	NULL	1	NULL	(working) ⇒ Need a further decision.
2	0	2	no	(not working AND income < \$50000) ⇒ DO NOT grant a loan.
3	0	3	NULL	(not working AND income ≥ \$50000) ⇒ Need a further decision.
4	1	8	no	(working AND credit rating = poor) ⇒ DO NOT grant a loan.
5	1	7	NULL	(working AND credit rating = fair) ⇒ Need a further decision.
6	1	6	yes	(working AND credit rating = excellent) ⇒ Grant a loan.
7	3	8	no	(not working AND income ≥ \$50000 AND credit rating = poor) ⇒ DO NOT grant a loan.
8	3	7	no	(not working AND income ≥ \$50000 AND credit rating = fair) ⇒ DO NOT grant a loan.
9	3	6	yes	(not working AND income ≥ \$50000 AND credit rating = excellent) ⇒ Grant a loan.
10	5	4	no	(working AND credit rating = fair AND income < \$30000) ⇒ DO NOT grant a loan.
11	5	5	yes	(working AND credit rating = fair AND income ≥ \$30000) ⇒ Grant a loan.

Table 4. Results from the load-decision query

Name	Loan-Application Decision
John	yes
Tom	no

4 Implementing and Emulating Lightweight Functions in ORDBMS

Lightweight function datatypes do not exist in any major relational database, but we can use polymorphic objects in ORDBMS to emulate their behavior. To our best knowledge, ORACLE 10g [6] and DB2 V8 [5] are the only two relational databases supporting dynamic method dispatch (i.e. polymorphism).

4.1 Execution Models of Dynamic Method Dispatch in ORDBMS

Because of inheritance and polymorphism, ORDBMS object types in a hierarchy can have multiple method implementations over the same signature. These method implementations belong to related object types in the hierarchy and their definitions are stored in system catalog. A child method implementation overrides ancestors'. When an object method is invoked at run time, the database execution engine must traverse the hierarchy in order to determine the most pertinent implementation for execution. After reviewing the ORACLE 10g and DB2 V8 manuals, we found that these two systems follow completely opposite approaches in determining which method implementation to invoke.

In ORACLE 10g [6], when an object method is invoked, the execution engine will first determine the type (called *current type*) of the callee object instance. If no method implementation is found in the current type, then the engine searches up the type hierarchy to locate an inherited method. By contrast, DB2 takes a top-down approach: its execution engine starts with the *root method* and moves down the type hierarchy to locate the so-called *most specific dispatchable method*.

ORACLE's bottom-up design reduces the overhead of searching for the most pertinent method implementation if most method invocations are on methods defined closer to the actual object type in the type hierarchy, while DB2's top-down approach favors the opposite pattern of method invocations. For table-driven programming, most method invocations happen in overriding methods of child object types, therefore, theoretically a bottom-up approach should take less time in locating the most pertinent method implementation.

4.2 Issues of Table-Driven Programming in ORDBMS

Using ORDBMS's dynamic method dispatch is a natural approach to implement table-driven programming in SQL. However, as indicated in section 2.2, this object-relational approach has some shortcomings that can limit its application in a table-driven architecture. These shortcomings can be summarized as issues of: *catalog scalability*, *code data integration*, and *meta data/object management*.

Most DBMS systems are sure making data scalability as one of the most important features, but this might not extend to database objects stored in the system catalog. Even if catalog tables are scalable as data tables, their schema design is fixed and might not be as efficient as a user-defined one for a specific application. Code and data in ORDBMS are managed using different mechanisms (DDL vs. DML), even though the Object-Oriented idea is to encapsulate code

and data in one entity. On the other hand, a *pure relational* approach treats code as data and they are matched together using relational operators. Essentially, ORDBMS objects are metadata stored in the system catalog. Oracle and DB2 implement recursive dispatch procedures in order to query a hierarchy of objects and methods for late-binding invocations. This approach may be sufficient for a small set of hand-crafted objects. However, following the true relational spirit and storing functions in data tables can make managing metadata and code just like managing data.

4.3 User-Defined Lightweight Function Processors

Besides using ORDBMS polymorphic methods, a user-defined external processor can evaluate lightweight functions written in any language. Lightweight functions are stored as VARCHAR in tables. In a query, a processor calls an external evaluator to run these functions on the fly. These external processors can implement many scripts or programs written in, for example, R, SAS, Perl, or Python, etc. They can even process SQL using a DBMS dynamic SQL API.

5 Conclusions

In this paper, we have described a *Lightweight Model Base*: a table-driven methodology for storing, managing, and evaluating data mining or business models. It can be implemented with only an extended relational database, storing models as what we call *lightweight functions*, which are novel SQL function datatypes. They can be straightforward textual representations of function values.

We have also discussed *table-driven modeling*. In this idea, the introduction of lightweight models in relational databases is a way to integrate the software-engineering methodology of table-driven programming. This methodology advocates storing functions in tables. The model evaluation process is just a mechanical evaluation of “joined” data and functions. It would make stored business logic transparent for understanding and maintenance as relational data. As examples, we gave detailed accounts of how this methodology can be applied on models involving decision trees.

In fact, many applications use models that admit this approach. Business applications in particular, which can have a very large number of special-case rules or business logic, are suitable for development with lightweight functions. Examples discussed in this paper include forecasting and loan-application systems. This approach can be extended for other data mining algorithms, and ultimately should be useful in development of model base management systems.

References

1. P. P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. In D. S. Kerr, editor, *VLDB 1975*, page 173. ACM, 1975.
2. A. Eisenberg and J. Melton. SQL: 1999, Formerly Known as SQL 3. *SIGMOD Record*, 28(1):131–138, 1999.

3. J. Gray et al. The Lowell Report. In *SIGMOD 2003*, page 680. ACM, 2003.
4. Hung-chih Yang and Douglas Stott Parker Jr. Table-Driven Programming in SQL for Enterprise Information Systems. In *ICEIS*, pages 424–427, 2005.
5. IBM et al. *IBM DB2 Universal Database SQL Reference Volume 1 Version 8*. IBM, 2002.
6. ORACLE et al. *Oracle Database Application Developer's Guide - Object-Relational Features 10g Release 1 (10.1)*. Oracle, 2003.
7. J. R. Quilan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.
8. S. E. Smylie et al. Introducing Data Administration into a Business Organization. In S. T. March, editor, *ER 1987*, pages 47–51. North-Holland, 1987.

An Efficient Index Lattice for XML Query Evaluation

Wilfred Ng and James Cheng

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology, Hong Kong
`{wilfred, csjames}@cse.ust.hk`

Abstract. We have defined an XML structural index called the *Structure Index Tree (SIT)*, which eliminates duplicate structures arising from the equivalent subtrees in an XML document by merging them into a concise structure. In this paper, we impose a lattice structure on the SIT and call the structure a *SIT-lattice* in order to enhance the applicability of the index. A *SIT-Lattice Element (SLE)* is an index of an arbitrary subset of paths in the document. Since paths represent the structure of the XML data and each text node is associated with a unique path, we can define an SLE to filter out both irrelevant structures and text nodes. We demonstrate that SLEs are able to support effective querying over very large XML documents in memory-limited hand-held devices.

1 Introduction

It is well recognized that establishing an efficient index to aid in processing queries on XML data is important, for example, Dataguides [4], 1-index [11], A(k)-indexes [7], D(k)-indexes [2], M(k)-indexes [5], and F&B-index [6]. However, the use of a structural index to process value-based query conditions and structural path expressions is mainly hindered by two factors that are related to the size of the index: (1) huge *structure size* and (2) huge *extent size*. By structure size, we refer to the total number of nodes in the index. By extent size, we refer to, depending on whether we are addressing a node in the index or the index itself, either the number of equivalent nodes represented by the extent of the index node or the sum of the extent sizes of all the nodes in the index.

In this paper, we study the problems arising from these two factors and propose a solution by utilizing a lattice structure defined on an XML structural index, called the *Structure Index Tree* (or the *SIT* in short) [3]. The SIT has been introduced in our preliminary work [3] to aid in efficient evaluation of XPath queries on compressed XML data. The SIT is constructed based on the partitioning of paths in an XML document, while an element in the lattice is the index of an arbitrary subset of paths in the document. We call the lattice the *SIT-lattice* and its element a *SIT-lattice element*, or an *SLE* for short.

How do we address the structure size problem? We consider different combinations of the *root-to-leaf* paths in the SIT. In total, there are 2^n combinations, where n is the number of leaf nodes in the SIT, and each combination constitutes

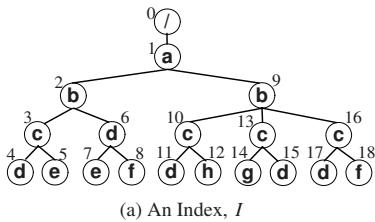


Fig. 1. A Full XML Index and a Lattice Element

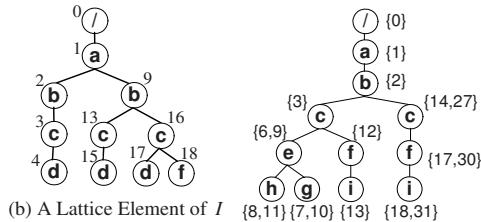


Fig. 2. A Sample SLE

an SLE. Therefore, the structure size of an SLE ranges from as small as the size of a single path to that of the full index, i.e., the SIT, which is the top of the index lattice. Compared with Kaushik et al.'s index [6] definition scheme and other indexing techniques [4], our proposal of using SLEs is much more flexible and effective, since we select the index of an arbitrary combination of paths that are relevant for query evaluation.

Example 1. Consider a full index, I , of an XML document, as shown in Figure 1(a). Suppose that we are only interested in the information of the elements “d” and “f” that are the children of “c” but not the siblings of “h”. To evaluate a query of this information, our method uses the XPath 2.0 union expression, “//c[not h]/(d | f)”, to specify an SLE and extract it from I , as depicted in Figure 1(b). With Kaushik et al.’s method, the minimal coverage is to select only the elements “c”, “d”, “f” and “h” and then check a “c” element by examining if it has a child, “h”. However, this is bound to be less efficient, since not only extra processing of the predicate is needed, but the resulting index also includes nodes such as “c₁₀” (node c with identity = 10), “d₆”, “d₁₁”, “f₈” and “h₁₂” which are irrelevant in the evaluation of a query of the required information.

How do we address the huge extent size problem? Consider an XML document that has 10,000 “a” elements and an A(k_L)-index that condenses the 10,000 nodes into 10 nodes, each having an extent size of 1,000. If an A(k_s)-Index, for some $k_s < k_L$, further condenses the 10 nodes into a single node, then the extent size of this single node will be increased to 10,000. Although the reduction in the structure size (from 10 nodes to 1 node) accelerates the evaluation of structural queries, such as “//a”, for a value-based query condition, such as “//x[a = ‘some value’]”, we have to match “some value” with the data value of each of the 10,000 “a” elements, even though there are few matches.

Our SIT-lattice is a well-defined structure that allows us to select only the relevant subset of nodes from the extent of an index node, since the SLE can select an arbitrary subset of paths from an XML document. We illustrate this idea of using the SLEs to accelerate query evaluation by the following example.

Example 2. Consider an XML document tree in Figure 3, where the attached integer of each node is its *node id*. Suppose we are only interested in the information related to the elements, “g”, “h” and “i”, that are descendants of a

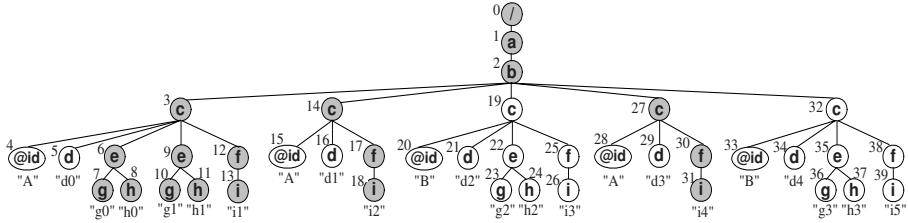


Fig. 3. An XML Document Tree

“c” element that has an “id” attribute of type “A”. To evaluate queries that retrieve data of these elements, such as “//c[@id = ‘‘A’’]//h”, we need only to access the shaded nodes in Figure 3. As mentioned before, we select a (any) combination of paths in an XML document and the resultant SLE is a very small index for the selected path. The SLE selected for our example is shown in Figure 2, which is an index of the shaded nodes in Figure 3. The SLE also pre-computes the common predicate “[@id = ‘‘A’’]” of the query workload.

In Figure 2, we can further combine the two equivalent paths, $\langle c, f, i \rangle$, into one; however, the collapsed index does not cover branching path expressions. For example, consider the query “//c[e]/f”. The “f” elements are not distinguishable with the two paths combined, but are distinguishable with the SLE in Figure 2. In fact, we find that the main factor that accelerates query evaluation is the reduction in the extent size, rather than further reduction in the structure size obtained by the coalescence of the two paths.

A practical problem arising from huge extent size is that in most cases the extents are too large to be loaded in the main memory of a machine. If we store the extents in a relational database then it incurs substantial disk I/O, resulting in degraded query performance. Our method partitions the full index into a set of SLEs, each of which can fit into the main memory. This approach is feasible in practice, since we usually access only a portion of the full index at any time. We make two main advancements on the SIT [3] in this paper.

First, we propose a novel lattice structure on the SIT. The lattice elements can effectively filter out irrelevant elements to accelerate query evaluation. Our method is efficient to tackle the problem of both the structure size and the extent size of an index on XML data. Second, we evaluate the SLEs on several benchmark datasets and a comprehensive set of queries. The results show that significant performance improvement is obtained and that using SLEs, we can efficiently query large XML datasets in a pocket-PC. Compared with Kaushik et al.’s index definition scheme [6], the SLEs are much easier and less costly to construct and more effective in controlling both the structure size and the extent size of an XML index.

In the rest of the section, we discuss the related work. We define the SIT-lattice and its related operations in Section 2. We evaluate the performance of the SLEs in Section 3. Finally, we give our concluding remarks in Section 4.

1.1 Related Work

A considerable amount of research has been conducted on indexing XML or semi-structured data [4][11][7][6][25]. However, none of the work has attempted to speed up the evaluation of value-based query conditions, which is crucial in querying XML data. We have discussed the A(k)-indexes [7], D(k)-indexes [2], M(k)-indexes [5], and the index definition scheme [6] to reduce the structure size of an index in Section 1. However, a new index of smaller structure size must be constructed from the base data, while the SLEs can be very efficiently constructed from existing SLEs by a set of lattice operations.

Marian et al. [8] constructs a projected document from a set of paths extracted from a given XQuery to reduce memory requirement for query processing. Their method works on the original XML document instead of an index. As the projected document in [8] is constructed from simple XPath expressions without predicates, the irrelevant nodes of value-based conditions are not filtered out. Buneman et al. [1] also proposes a lattice structure, which is defined on a class of equivalent *tree instances* based on bisimulation. However, they do not focus on constructing a lattice element of smaller size from existing lattice elements to accelerate query evaluation.

2 An Index Lattice

2.1 The XML Structure Index Tree (SIT)

The SIT is an index defined on the structure of XML data. We model an XML document as a tree, called the *structure tree*, $T = (V_T, E_T, \text{root}_T)$, where V_T and E_T are the sets of tree nodes and edges in T , respectively, and root_T is the unique root of T . Each edge in E_T specifies the parent-child relationship of two nodes. Each tree node, $v \in V_T$, is defined as $v = (lid, nid, ext)$, where $v.lid$ is the unique identifier of the element/attribute label generated by a hash function; $v.nid$ is the unique node identifier assigned to v according to the document order; and ext denotes the *extent* associated with v , which contains the *nids* of the set of equivalent nodes that are coalesced into v . We set $v.ext = \{v.nid\}$ (i.e. $v.ext$ is a singleton), which is later to be combined with the *exts* of other equivalent nodes to obtain the SIT.

Each v is identified by the $(v.lid, v.nid)$ pair and the identity of root_T is uniquely assigned to be $(0, 0)$. In addition, if v has n children $(\beta_1, \dots, \beta_n)$, their order is specified as: (1) $\beta_1.lid \leq \beta_2.lid \leq \dots \leq \beta_n.lid$; and (2) if $\beta_i.lid = \beta_{i+1}.lid$, then $\beta_i.nid < \beta_{i+1}.nid$. This node ordering accelerates node selection in T by an approximate factor of 2, since we match the nodes by their *lids* and, on average, we only need to search half of the children of a node in T . As an example, Figure 4 shows the structure tree of the XML document in Figure 3.

Each text node in an XML document is attached to a unique path, p , in the structure tree, which is given by $p = v_0v_1 \dots v_n$, where v_n is a leaf node. We take into account the numerical order of *lid* and *nid* and define a path ordering as follows.

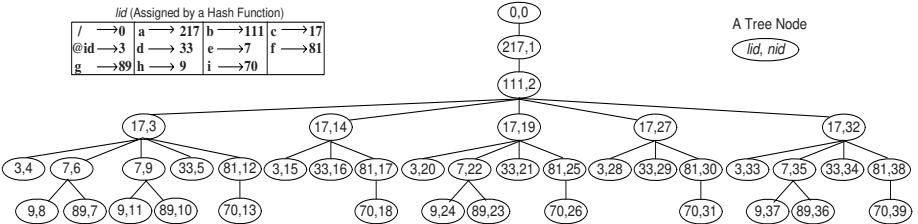


Fig. 4. The Structure Tree of the XML document presented in Figure 3

Definition 1. (Path Ordering) Given two paths, $p_1 = u_0 \dots u_m$ and $p_2 = v_0 \dots v_n$, $p_1 \preceq p_2$ if one of the following two conditions holds:

1. $p_1 \prec p_2$: there exists some i , where $0 \leq i < \min(m, n)$, such that $u_i.nid = v_i.nid$ and $u_{i+1}.nid \neq v_{i+1}.nid$, and
 - 1.1 $u_{i+1}.lid < v_{i+1}.lid$; or
 - 1.2 $u_{i+1}.lid = v_{i+1}.lid$ and $u_{i+1}.nid < v_{i+1}.nid$.
2. $p_1 = p_2$: $u_i.nid = v_i.nid$, for $0 \leq i \leq m$ and $m = n$.

With the path ordering, we can specify a structure tree (or a structure subtree), T , as the set of all its paths ordered as follows: $T = p_0 \preceq \dots \preceq p_n$ (or simply $T = p_0 \prec \dots \prec p_n$ as the paths are distinct in T). To eliminate duplicate structures in a structure tree, we introduce the notion of *SIT-equivalence*, which is employed to merge duplicate paths and subtrees to obtain the SIT.

Definition 2. (SIT-equivalence) Two paths, $p_1 = u_0 \dots u_m$ and $p_2 = v_0 \dots v_n$, are *SIT-equivalent*, if $u_i.lid = v_i.lid$ for $0 \leq i \leq m$ and $m = n$. Two subtrees, $T_1 = p_{10} \preceq \dots \preceq p_{1m'}$ and $T_2 = p_{20} \preceq \dots \preceq p_{2n'}$, are *SIT-equivalent*, if (1) the roots of T_1 and T_2 are siblings and (2) p_{1i} and p_{2i} are SIT-equivalent for $0 \leq i \leq m'$ and $m' = n'$.

The following example helps illustrate the concepts of branch ordering and SIT-equivalence.

Example 3. Given $p_1 = "(0,0) \dots (3,4)"$, $p_2 = "(0,0) \dots (9,8)"$ and $p_3 = "(0,0) \dots (3,15)"$ in Figure 4, and $p_4 = "(0,0) \dots (3,15)"$ in Figure 5, we have $p_1 \prec p_2 \prec p_3$ and $p_3 = p_4$. The subtrees rooted at the nodes (17,14) and (17,27) in Figure 4 are SIT-equivalent, since every pair of corresponding paths in these two subtrees are SIT-equivalent. The subtrees rooted at the nodes (17,19) and (17,32) are also SIT-equivalent.

Since the structures of SIT-equivalent subtrees are duplicate, we define a tree-merge operation (cf. [3]) to eliminate the redundant tree structures by merging the SIT-equivalent subtrees, T_1 and T_2 . We skip the details of the algorithm but only give an example of the operation here: if we apply the *merge* operation to the SIT-equivalent subtrees rooted at the nodes (17,14) and (17,27) in Figure 4, the resultant merged subtree is the subtree rooted at (17,14) in Figure 5.

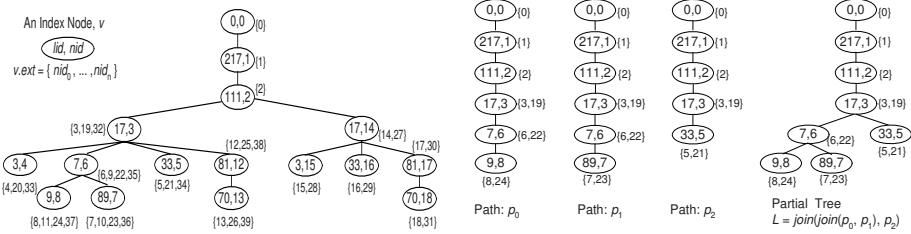


Fig. 5. The SIT of the XML Document in Figure 3

Fig. 6. A Partial SIT Constructed by Joining Three Paths, p_0, p_1 and p_2

2.2 The SIT-Lattice

Given a set of paths, $P = \{p_0, \dots, p_k\}$, in the SIT, we define the path-join operation, *join*, as shown in Procedure 1, which joins the paths in P one by one to obtain a partial tree.

Procedure 1. $join(L, p)$

/ $L = p_0 \preceq \dots \preceq p_{k-1}$ and $p_{k-1} \preceq p_k$, where $p_{k-1} = u_0 \dots u_m$ and $p_k = v_0 \dots v_n$ */*

begin

- ```

1. for each $0 \leq i \leq m$ do
2. if ($u_i.nid = v_i.nid$) then
3. $u_i.ext := u_i.ext \cup v_i.ext$;
4. Delete v_i and its outgoing edge, if any;
5. else
6. Connect $v_i \dots v_n$ to T such that v_i is the last child of u_{i-1} ;
7. return L ;
8. return L ;
end

```

We can apply *join* on a set of selected paths to obtain a tree, which we call a *partial SIT*, as defined in Definition B.

**Definition 3. (Partial SIT)** Let  $P = \{p_0, p_1, \dots, p_k\}$  be a set of paths in the SIT. Without loss of generality, we assume that  $p_0 \preceq p_1 \preceq \dots \preceq p_k$ . A *Partial SIT*,  $L$ , over  $P$ , is a tree constructed as follows:  $L = \text{join}(\dots \text{join}(L', p_1), \dots, p_k)$ , where  $L'$  is the initial tree that consists of only one path,  $p_0$ .

*Example 4.* If we apply the *join* operation to the three paths,  $p_0$ ,  $p_1$  and  $p_2$ , in Figure 6, we obtain the partial SIT,  $L = \text{join}(\text{join}(p_0, p_1), p_2)$ . Note that the paths are joined together by the SIT-equivalent portions of the paths.

Each path in the SIT is the concise representation of a set of SIT-equivalent paths,  $P_T$ , in the structure tree,  $T$ . However, in most cases, only a subset of  $P_T$  is useful for the evaluation of a given query workload. We define an *index path* that concisely represents any subset of  $P_T$ .

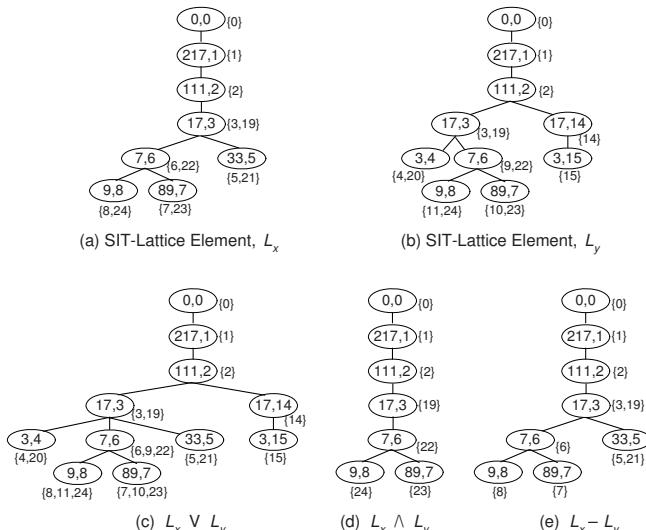
**Definition 4. (Index Path)** Let  $P_T$  be the set of all paths in a structure tree represented by a path in its SIT and  $p \in P_T$ ,  $p = u_0 \dots u_n$ . An *index path*,  $p_I = v_0 \dots v_n$ , is a path in a partial SIT such that  $v_i.nid = u_i.nid$ ,  $v_i.lid = u_i.lid$ , and  $v_i.ext = \bigcup_{\exists p \in P_T} \{u_i.nid\}$ , for  $0 \leq i \leq n$ .

*Example 5.* The three paths in Figure 6 are index paths of some partial SIT. For example,  $p_0$  represents the two paths, “ $(0, 0) \dots (9, 8)$ ” and “ $(0, 0) \dots (9, 24)$ ”, in Figure 4 and its corresponding index path in the SIT is the path “ $(0, 0) \dots (9, 8)$ ” shown in Figure 5.

**Theorem 1.** The set of all partial SITs defined over a SIT is a lattice.

We call this lattice defined over the SIT the *SIT-lattice* and an element in the SIT-lattice, i.e., a partial SIT, a *SIT-lattice element* or simply an *SLE*. Therefore, the *maximum SLE* is the SIT and the *minimum SLE* is an empty tree. The least upper bound and the greatest lower bound of two SLEs,  $L_x$  and  $L_y$ , i.e.  $(L_x \vee L_y)$  and  $(L_x \wedge L_y)$ , are also referred to as the *union* and the *intersection* of  $L_x$  and  $L_y$ , respectively. To allow more flexible construction of useful SLEs to aid query evaluation, we introduce two more SIT-lattice operations, *subtraction* and *extraction*. The subtraction of two SLEs,  $(L_x - L_y)$ , is the index of the set of paths  $P = (P_x - P_y)$ , where  $P_x$  and  $P_y$  are the set of paths indexed by  $L_x$  and  $L_y$  respectively. We say  $L_x$  is an extraction of  $L_y$  if  $L_x \leq L_y$ .

*Example 6.* Figure 7 shows two SLEs,  $L_x$  and  $L_y$ , and their union ( $L_x \vee L_y$ ), intersection ( $L_x \wedge L_y$ ) and subtraction ( $L_x - L_y$ ). All the five SLEs are extractions of the SIT in Figure 5, while  $(L_x - L_y)$  is an extraction of  $L_x$  and  $(L_x \wedge L_y)$  is an extraction of  $L_x$  (or  $L_y$ ), which in turn is an extraction of  $(L_x \vee L_y)$ .



**Fig. 7.** SIT-lattice Elements and Operations

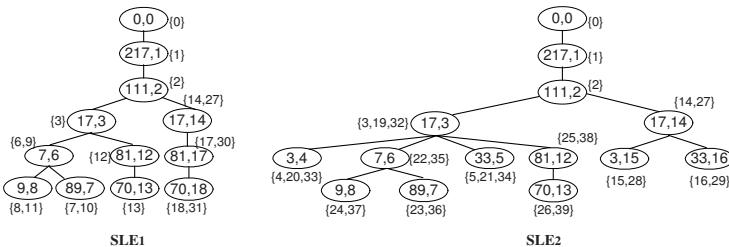
### 2.3 Heuristic Selection Rules

The problem of specifying an SLE,  $L$ , to cover a given set of queries,  $Q$ , is equivalent to checking whether the set of nodes selected by  $L$  is a superset of the union of the set of nodes selected by  $q \in Q$ . We call this problem the SLE containment problem.

The containment problem for XPath fragments (c.f. A survey on XPath query containment [13]), that consist of the “*child*” axis and any two of the following three constructs, (1) “*descendant*” axis, (2) predicates, and (3) wildcards, is shown to be in PTIME in [11]. However, the containment problem for the XPath fragment that consists of all three constructs is shown to be co-NP complete [9], while adding the union expression “|” to the fragment makes the containment problem to be in EXPTIME [12].

The SLE containment problem is even harder, since we allow a richer set of XPath features such as aggregation-based and value-based predicates. Therefore, we employ a set of heuristic rules to aid the specification of an efficient SLE. For example, given the three queries, “//a/b/c”, “//a/b/d//e” and “//a/b/d//f”, we can specify an SLE to cover the queries as  $L = “//a/b/(c \mid d//(e \mid f))”$ , or simply some less-efficient upper bounds of  $L$ , such as “//a/b/(c \mid d)” and “//a/b”. We skip the details of our rules due to space limitation.

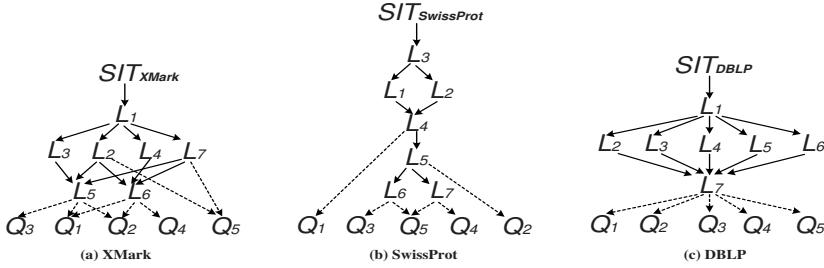
The indexes of real XML datasets [10] are often too large to be loaded into the main memory of a machine, especially hand-held devices such as pocket-PCs. Apart from extracting an SLE from a large index to reduce the index size, we can also partition a large index into smaller partitions in order to load them into the main memory. For example, “//c[.//d >= 10]/(e \mid f)” partitions the SIT in Figure 5 into two SLEs, as shown in Figure 8.



**Fig. 8.** Horizontal Partition of the SIT in Figure 5

## 3 Experimental Evaluation

We carried out two sets of experiments. The first is on a Windows XP machine with a P4, 2.53 GHz processor and 512 MB of RAM. The second is to use a Toshiba Pocket-PC with a 400 MHz Intel PXA250 processor and 64 MB of SDRAM; we loaded the SLEs in the Pocket-PC’s main memory and retrieved the data contents of the result nodes from the PC via a wireless LAN with a transfer rate of 11 Mbps. We used the following three datasets [10] XMark, SwissProt

**Fig. 9.** SIT-Lattice Elements and Queries

and DBLP. We list the queries ( $Q_1$  to  $Q_5$ ) and the SLEs ( $L_1$  to  $L_7$ ) in Appendix [4], while we depict an overview of the relationships between the SLEs and the queries for each dataset in Figure 9. In the figure, a (dotted) path from an SLE,  $L_i$ , to a query,  $Q_j$ , means that  $L_i$  covers  $Q_j$ , while a (solid) path from an SLE,  $L_i$ , to another SLE,  $L_j$ , indicates that  $L_j \leq L_i$ . For simplicity, we use  $L_{i,\dots,j}$  to denote  $L_i, \dots, L_j$  in subsequent discussions.

### 3.1 Effectiveness of Using SLEs

**Performance on SLE Construction.** We investigate (1) the effectiveness of the SLEs in controlling the structure size and the extent size of the index and (2) the efficiency in constructing the SLEs. In Table 1 we show the Structure Ratio and Extent Ratio of the SLEs of the three XML datasets,  $L_1$  to  $L_7$ , which represent the ratio of the structure size and the extent size of the respective SLEs to those of their corresponding SIT, respectively.

The results show that the structure size and the extent size of the SLEs can essentially vary from as small as 0% to as large as 100% of the SIT, and many points in between. This implies that we have great flexibility in choosing an SLE to aid in query evaluation.

We also record the time (Build Time) taken to construct the SLEs in Table 1. The Build Time includes the time taken to load the SLE into the main memory, though the loading time is usually negligible compared to the construction time. When the SLEs (such as  $L_{1,2,3,4}$  of XMark,  $L_{1,2,5,6,7}$  of SwissProt and  $L_{1,2,3,4,5,6}$

**Table 1.** SLE Construction Results

| SIT-Lattice Elements |                     | $L_1$ | $L_2$ | $L_3$ | $L_4$ | $L_5$ | $L_6$ | $L_7$ |
|----------------------|---------------------|-------|-------|-------|-------|-------|-------|-------|
| XMark                | Structure Ratio (%) | 11.98 | 0.84  | 4.95  | 7.91  | 0.31  | 0.40  | 0.58  |
|                      | Extent Ratio (%)    | 34.18 | 0.59  | 6.04  | 16.42 | 0.41  | 0.43  | 0.69  |
|                      | Build Time (sec)    | 0.233 | 1.231 | 1.032 | 1.520 | 0.001 | 0.001 | 0.011 |
| SwissProt            | Structure Ratio (%) | 81.11 | 57.80 | 88.43 | 42.95 | 35.67 | 22.56 | 31.81 |
|                      | Extent Ratio (%)    | 79.48 | 59.33 | 90.60 | 45.28 | 37.20 | 23.79 | 33.12 |
|                      | Build Time (sec)    | 5.123 | 7.020 | 0.078 | 0.021 | 0.230 | 0.167 | 0.188 |
| DBLP                 | Structure Ratio (%) | 22.96 | 10.34 | 11.72 | 9.16  | 7.25  | 8.58  | 0.64  |
|                      | Extent Ratio (%)    | 54.23 | 2.32  | 13.39 | 2.54  | 1.13  | 0.16  | 0.001 |
|                      | Build Time (sec)    | 0.560 | 1.709 | 1.121 | 1.530 | 1.002 | 1.402 | 0.044 |

of DBLP) are extracted from their upper bounds, it is usually more costly if value-based predicates are imposed, since we need to access the disk to retrieve the data contents of the nodes for the evaluation of the predicates. However, when the SLEs (such as  $L_{5,6,7}$  of XMark,  $L_{3,4}$  of SwissProt and  $L_7$  of DBLP) are constructed as the union or the intersection of some existing SLEs, the construction time is only on average tens of milliseconds.

**Query Evaluation Speedup.** We study the query evaluation speedup obtained by using the SLEs instead of the SITs. Our goal is to investigate the effect of a reduction in the structure size and/or the extent size on the query performance. We measure the response time of each query that is evaluated using the SLEs and the SIT. Then, we compute the speedup as the ratio of the response time of a query evaluated using an SLE to that using the SIT. We show the speedup ratio (*milliseconds per second*) in Table 2. For example, for XMark, the speedup ratio of  $L_4$  against  $Q_1$  is 80, which means that it takes 80 milliseconds to evaluate  $Q_1$  using  $L_4$ , while it takes 1 second to evaluate  $Q_1$  using the SIT. Thus, a lower speedup ratio indicates a higher speedup. In Table 2, a slash “/” indicates that the SLE does not cover the query. We record impressive speedup for all the three datasets and thus no speedup ratio is presented here.

Based on the experimental results, we derive a guideline to achieve better query performance using SLEs: more emphasis should be put on reducing the extent size (by imposing value-based predicates) than on reducing the structure size (by imposing structural predicates). However, we note that for less regular data sources, such as SwissProt, reducing the structure size and reducing the extent size are equally important, because it is likely that every index node is associated with only a few elements. For such datasets, it is more effective to reduce the structure size, since a reduction in the structure size also effectively brings down the extent size of the index, as shown by SwissProt.

Finally, we remark that in this experiment, we evaluated all the predicates imposed on the queries, even though part of them are already pre-computed by the SLEs. The reason for the re-computation is to give an accurate account of

**Table 2.** Query Evaluation Speedup Ratio (msec/sec)

| SIT-Lattice Elements | $L_1$ | $L_2$ | $L_3$ | $L_4$ | $L_5$ | $L_6$ | $L_7$ |
|----------------------|-------|-------|-------|-------|-------|-------|-------|
| XMark                | $Q_1$ | 933   | 103   | 147   | 80    | 10    | 7     |
|                      | $Q_2$ | 912   | 138   | 212   | 96    | 17    | 9     |
|                      | $Q_3$ | 986   | 33    | 46    | /     | 9     | 24    |
|                      | $Q_4$ | 877   | 35    | /     | 41    | /     | 18    |
|                      | $Q_5$ | 987   | 93    | /     | /     | /     | 19    |
|                      | $Q_6$ | 356   | 171   | 836   | 41    | /     | /     |
| SwissProt            | $Q_1$ | 334   | 194   | 719   | 71    | 33    | /     |
|                      | $Q_2$ | 455   | 310   | 987   | 112   | 133   | 87    |
|                      | $Q_3$ | 519   | 441   | 1031  | 106   | 118   | /     |
|                      | $Q_4$ | 414   | 426   | 761   | 209   | 126   | 108   |
|                      | $Q_5$ | 904   | 92    | 537   | 123   | 45    | 19    |
|                      | $Q_6$ | 810   | 64    | 424   | 60    | 26    | 10    |
| DBLP                 | $Q_1$ | 940   | 159   | 577   | 219   | 83    | 52    |
|                      | $Q_2$ | 1034  | 88    | 243   | 107   | 37    | 35    |
|                      | $Q_3$ | 911   | 146   | 751   | 128   | 69    | 27    |
|                      | $Q_4$ | 904   | 92    | 537   | 123   | 45    | 19    |
|                      | $Q_5$ | 810   | 64    | 424   | 60    | 26    | 10    |

the effects of the reduction in the structure size and the extent size on query performance. However, it is interesting to see that when we made use of predicate pre-computation, significantly greater speedup was measured for almost all of the SLEs. In real-world database applications, a user can take advantage of this feature of the SLE to obtain efficient query performance gain.

**Query Performance Gain.** We now measure the gain in query performance obtained by using the SLEs instead of the SIT and then illustrate the applicability of the SLEs by an example. We measure the performance gain as  $(1 - (SLE\ Construction\ Cost + Query\ Evaluation\ Cost\ using\ the\ SLE) / Query\ Evaluation\ Cost\ using\ the\ SIT)$ , i.e.,  $G = \{1 - (c_l + \sum_{i=1}^n c'_i) / \sum_{i=1}^n c_i\} \times 100\%$ , where  $c_i$  and  $c'_i$  are the costs of evaluating the  $i$ th query in the workload using the SIT and the SLE, respectively, and  $c_l$  is the cost of building the SLE. We present in Table 3 the percentage gains for two scenarios:  $G+$  reports the gain of using an SLE assuming that the SLE was constructed from some existing SLEs other than the SIT, while  $G-$  reports the gain of an SLE that was constructed (all the way) from the SIT. For example, the construction cost of  $L_7$  of XMark is 0.011 second, as reported in Table 1, for the  $G+$  scenario. However, the cost is 4.029 secs, which is the sum of the construction time of all the seven SLEs, for the  $G-$  scenario, since all other SLEs must be constructed before  $L_7$  can be constructed.

**Table 3.** Query Performance Gain

| SIT-Lattice Elements |          | $L_1$ | $L_2$ | $L_3$ | $L_4$ | $L_5$ | $L_6$ | $L_7$ |
|----------------------|----------|-------|-------|-------|-------|-------|-------|-------|
| XMark                | $G+$ (%) | 4.06  | 86.43 | 76.89 | 79.99 | 98.74 | 98.94 | 97.76 |
|                      | $G-$ (%) | 4.06  | 85.53 | 74.95 | 78.08 | 77.93 | 74.46 | 82.10 |
| SwissProt            | $G+$ (%) | 55.43 | 63.70 | 12.66 | 87.73 | 89.06 | 90.02 | 90.43 |
|                      | $G-$ (%) | 55.43 | 63.70 | 8.26  | 83.34 | 84.07 | 80.87 | 81.63 |
| DBLP                 | $G+$ (%) | 7.60  | 89.33 | 53.31 | 88.11 | 95.00 | 96.73 | 99.98 |
|                      | $G-$ (%) | 7.60  | 89.04 | 53.03 | 87.82 | 94.71 | 96.45 | 96.26 |

On average, using the SLEs instead of the SIT achieves significant improvement in query evaluation in both scenarios. The percentage gain is over 70% for most of SLEs, in both  $G+$  and  $G-$  scenarios. The small difference between  $G+$  and  $G-$  also implies the great efficiency in constructing the SLEs. Those less obvious performance gains shown in Table 3 can be explained by the small query evaluation speedup measured for these SLEs. This is also because we only used 5 queries for each SLE in this experiment. In practice, more queries are generally posed at a given time and the performance gain can still be further increased.

### 3.2 Use of SLEs in Memory-Limited Devices

The goal of this experiment is to show that the SLEs allow efficient querying of large XML data in memory-limited devices. We partition XMark and construct an SLE for each child of the root of its SIT. We horizontally partition SwissProt into 12 SLEs of roughly the same size by specifying each SLE as “//Entry[@seqlen

[.  $\leq range\_lower$  and .  $\geq range\_upper$ ].” For DBLP, we first apply Vertical Partition by constructing an SLE for each child of the root of the SIT of DBLP and then horizontally partition the over-sized child “*inproceedings*” as “// *inproceedings*[@key starts-with ‘‘conf/somevalue/’’]”. Using the partition strategies, the indexes of all the three datasets are able to be loaded into the main memory of the pocket-PC. Note that the SLEs are constructed from their corresponding SITs in the PC machine, since the SITs are too large to be loaded into the main memory of the pocket-PC.

To assess the query performance, we construct, in the pocket-PC,  $L_{2,3,4,5,6,7}$  (c.f. Appendix [14]) from  $L_1$  for XMark and DBLP. However,  $L_1$  of DBLP is too large to be loaded into the main memory of the pocket-PC. We thus horizontally partition  $L_1$  of DBLP into four SLEs:  $L_{11}$ ,  $L_{12}$ ,  $L_{13}$  and  $L_{14}$ . Then, we extract  $L_{2j,3j,4j,5j,6j}$  from  $L_{1j}$  and construct  $L_{7j}$  as the intersection of  $L_{2j,3j,4j,5j,6j}$ , where  $j$  is 1, 2, 3 and 4, respectively. Finally,  $L_i$  of DBLP is constructed as the union of  $L_{i1,i2,i3,i4}$  for  $2 \leq i \leq 7$  and then loaded into the pocket-PC. Then, we evaluate the same set of queries (c.f. Appendix [14]) by using the SLEs. We measure the speedup ratio as the ratio of the response time of evaluating a query using an SLE to that using  $L_1$ . The query performance gains that we obtain for each of the SLEs are on average slightly better than but roughly of the same pattern as those obtained on the PC machine as shown in Sections B.1 (detailed experimental results thus omitted).

## 4 Conclusions

We have presented the SIT-lattice defined on the SIT. With the SIT-lattice, we are able to select any subset of relevant paths from an XML document. A SIT-lattice element (SLE) is specified by an XPath expression.

We carried out empirical studies of SLEs as follows. First, we showed with experimental evidence that the SLEs can be constructed very efficiently and that using the SLEs, instead of the full index, can tremendously improve query performance. Second, we demonstrated that SLEs can be used to query large XML data with impressive query performance in Pocket-PCs.

We remark that, in general, it is difficult to check whether an SLE fully covers a given query workload, as studied in the containment problem of XPath fragments in [11][9][12]. However, in a distributed environment, such as using hand-held devices in a P2P network, it is important for users to obtain a fast response of query results, despite the fact that the results may not be complete. In such environments, SLEs can not only be used as efficient query accelerators, but can also be used to partition the indexes to allow them to fit into the main memory of the memory-limited devices.

## References

1. P. Buneman, et al. Path Queries on Compressed XML. In *Proc. of VLDB*, 2003.
2. Q. Chen, A. Lim, and K. W. Ong. D(K)-Index: An Adaptive Structural Summary for Graph-Structured Data. In *Proceedings of SIGMOD*, 2003.

3. J. Cheng and W. Ng. XQzip: Querying Compressed XML Using Structural Indexing. In *Proceedings of EDBT*, 2004.
4. R. Goldman and J. Widom. Dataguides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of VLDB*, 1997.
5. H. He and J. Yang. Multiresolution Indexing of XML for Frequent Queries. In *Proceedings of ICDE*, 2004.
6. R. Kaushik, P. Bohannon, J. F. Naughton and H. F. Korth. Covering Indexes for Branching Path Queries. In *Proceedings of SIGMOD*, 2002.
7. R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting Local Similarity for Efficient Indexing of Paths in Graph Structured Data. In *Proceedings of ICDE*, 2002.
8. A. Marian and J. Simeon. Projecting XML Documents. In *Proc. of VLDB*, 2003.
9. G. Miklau and D. Suciu. Containment and Equivalence for a Fragment of XPath. In *Journal of the ACM*, Vol. 51, No. 1, pp.2-45, January 2004.
10. G. Miklau and D. Suciu. XML Data Repository, which can be found at the URL: <http://www.cs.washington.edu/research/xmldatasets>.
11. T. Milo and D. Suciu. Index Structures for Path Expressions. In *Proceedings of ICDT*, 1999.
12. F. Neven and T. Schwentick. XPath Containment in the Presence of Disjunction, DTDs, and Variables. In *Proceedings of ICDT*, 2003.
13. T. Schwentick. XPath Query Containment. In *SIGMOD Record*, 33(1), 2004.
14. Appendix <http://www.cse.ust.hk/~wilfred/SLE/appendix.pdf>.

## Appendix

(This appendix [14] is included for reading convenience only).

This appendix lists, in abbreviated XPath syntax, the queries and the specification of the SLEs used in the performance evaluation. We use fully parenthesized expressions for the predicates as to avoid ambiguity.

We use three benchmark XML datasets: XMark, which is an XML benchmark project modelling a deeply nested auction database; SwissProt, which describes DNA sequences; and DBLP, which is a popular bibliography database. Table 4 shows some brief descriptions of the three XML datasets such as the size, the number of distinct tags/attributes, and the maximum depth of each dataset.  $|V_T|$  is the number of nodes in the structure tree, which is the extent size of the SIT, and  $|V_I|$  is the number of nodes in the SIT, which is the structure size of the SIT. The ratio of  $|V_I|$  to  $|V_T|$  shown in the last column of Table 4 indicates the degree of its redundancy (a higher ratio indicates less redundancy) and regularity (a lower ratio indicates greater regularity) of the dataset. Thus, the  $|V_I|/|V_T|$  ratios show that DBLP is relatively regular and SwissProt has the lowest level of redundancy.

**Table 4.** Dataset Descriptions

| Datasets  | Size   | Tags/Attrs | Depth | $ V_T $ | $ V_I $ | $ V_I / V_T $ |
|-----------|--------|------------|-------|---------|---------|---------------|
| XMark     | 111 MB | 86         | 11    | 1837608 | 30071   | 1.64%         |
| SwissProt | 109 MB | 100        | 5     | 5166890 | 1466332 | 28.38%        |
| DBLP      | 127 MB | 38         | 5     | 3733320 | 1874    | 0.05%         |

## XMark

Common predicates used in the queries and the SLE specification:

$$\begin{aligned} P_{x1} &= [[[initial \geq 100] \text{ and } [current \leq 200]] \text{ and } [not [reserve]]] \\ P_{x2} &= [\text{interval}[[start \geq 01/01/2000] \text{ and } [end < 01/01/2001]]] \\ P_{x3} &= [[\text{count}(\text{bidder}) \geq 10] \text{ and } [\text{avg}(\text{bidder/increase}) < 5]] \end{aligned}$$

Queries:

$$\begin{aligned} Q_1 &: //\text{site}/\text{open_auctions}/\text{open_auction}[P_{x1} \text{ and } [P_{x2} \text{ and } P_{x3}]]/\text{@id} \\ Q_2 &: //\text{site}/\text{open_auctions}/\text{open_auction}[[P_{x1} \text{ and } [P_{x2} \text{ and } P_{x3}]] \text{ and } [not [bidder]]]/(\text{@id} \mid /*\text{/description}}) \\ Q_3 &: //\text{open_auction}[[P_{x1} \text{ and } P_{x2}]] \text{ and } [\text{type} = \text{'featured'}]/\text{@id} \\ Q_4 &: //\text{site}/\text{open_auctions}/\text{open_auction}[[P_{x1} \text{ and } P_{x3}]] \text{ and } [\max(\text{bidder/increase}) \geq 10]/\text{annotation}/\text{description} \\ Q_5 &: //\text{open_auction}[[P_{x1} \text{ and } [P_{x2} \text{ or } P_{x3}]] \text{ and } [not [\text{contains}(\text{type}, \text{'Dutch'})]]]/(\text{@id} \mid \text{bidder}[\text{increase} \geq 10]/\text{date}) \end{aligned}$$

SIT-lattice elements:

$$\begin{aligned} L_1 &: //\text{open_auctions} \\ L_2 &: //\text{open_auction}[P_{x1}]/(\text{@id} \mid /*\text{/description} \mid \text{type} \mid \text{bidder}/(\text{date} \mid \text{increase}) \mid \text{interval}) \\ L_3 &: //\text{open_auction}[P_{x2}] \\ L_4 &: //\text{open_auction}[P_{x3}] \\ L_5 &= L_2 \cap L_3: //\text{open_auction}[P_{x1} \text{ and } P_{x2}]/(\text{@id} \mid /*\text{/description} \mid \text{type} \mid \text{bidder}/(\text{date} \mid \text{increase}) \mid \text{interval}) \\ L_6 &= L_2 \cap L_4: //\text{open_auction}[P_{x1} \text{ and } P_{x3}]/(\text{@id} \mid /*\text{/description} \mid \text{type} \mid \text{bidder}/(\text{date} \mid \text{increase}) \mid \text{interval}) \\ L_7 &= L_5 \cup L_6: //\text{open_auction}[P_{x1} \text{ and } [P_{x2} \text{ or } P_{x3}]]/(\text{@id} \mid /*\text{/description} \mid \text{type} \mid \text{bidder}/(\text{date} \mid \text{increase}) \mid \text{interval}) \end{aligned}$$

## SwissProt

Common predicates used in the queries and the SLE specification:

$$\begin{aligned} P_{x1} &= [\text{@seqlen}[[. \geq 100] \text{ and } [. < 1000]]] \\ P_{x2} &= [\text{Mod}[[\text{@type} = \text{'Created'}] \text{ and } [\text{@date}[[. \geq \text{'01-JAN-1993'}] \text{ and } [. < \text{'1-JAN-2000'}]]]]] \\ P_{x3} &= [P_{x1} \text{ and } P_{x2}] \\ P_{x4} &= [P_{x3} \text{ and } [\text{count}(\text{Ref}) = 1]] \\ P_{x5} &= [P_{x4} \text{ and } [\text{contains}(\text{Species}, \text{'Homo'})]] \end{aligned}$$

Queries:

$$\begin{aligned} Q_1 &: //\text{Entry}[P_{x3}]/(\text{@id} \mid \text{Gene}) \\ Q_2 &: //\text{Entry}[P_{x4}]/(\text{@id} \mid \text{Gene}) \\ Q_3 &: //\text{Entry}[P_{x5} \text{ and } [\text{count}(\text{Keyword}) \geq 5]]/(\text{@id} \mid \text{Gene}) \\ Q_4 &: //\text{Entry}[P_{x5} \text{ and } [\text{count}(\text{Org}) \geq 5]]/(\text{@id} \mid \text{Gene}) \\ Q_5 &: //\text{Entry}[P_{x5} \text{ and } [[\text{count}(\text{Keyword}) \geq 5] \text{ and } [\text{count}(\text{Org}) \geq 5]]]/(\text{@id} \mid \text{Gene}) \end{aligned}$$

SIT-lattice elements:

$$\begin{aligned} L_1 &: //\text{Entry}[P_{x1}] \\ L_2 &: //\text{Entry}[P_{x2}] \\ L_3 &= L_1 \cup L_2: //\text{Entry}[P_{x1} \text{ or } P_{x2}] \\ L_4 &= L_1 \cap L_2: //\text{Entry}[P_{x3}] \\ L_5 &: //\text{Entry}[P_{x4}] \\ L_6 &: //\text{Entry}[P_{x4} \text{ and } [\text{count}(\text{Keyword}) \geq 5]] \\ L_7 &: //\text{Entry}[P_{x4} \text{ and } [\text{count}(\text{Org}) \geq 5]] \end{aligned}$$

## DBLP

Common predicates used in the queries and the SLE specification:

$P = [[[contains(\text{author}, \text{'David'})] \text{ and } [year \geq 2000]] \text{ and }$   
 $[\text{crossref}[[contains(\cdot, \text{'sigmod'})] \text{ or } [contains(\cdot, \text{'vldb'})]]]] \text{ and }$   
 $[contains(\text{booktitle}, \text{'SIGMOD'})]] \text{ and } [contains(\text{title}, \text{'Data Mining'})]]]$

Queries:

$Q_1: //*/@\text{key}[\text{ancestor-or-self}::\text{inproceedings}[P]]$

$Q_2: (\text{//title}[\text{parent}::\text{inproceedings}[P]] \mid$   
 $\text{//author}[\text{parent}::\text{inproceedings}[P]])$

$Q_3: //*/\text{inproceedings}[P]/(\text{booktitle} \mid \text{year} \mid \text{page} \mid \text{title})$

$Q_4: //\text{cite}[@\text{label}[\cdot = \text{'IBM99'} \text{ and } \cdot/\text{ancestor}::\text{inproceedings}[P]]]$

$Q_5: \text{count}(\text{//inproceedings}[P]/\text{author})$

SIT-lattice elements:

$L_1: //\text{inproceedings}$

$L_2: //\text{inproceedings}[contains(\text{author}, \text{'David'})]$

$L_3: //\text{inproceedings}[\text{year} \geq 2000]$

$L_4: //\text{inproceedings}[\text{crossref}[[contains(\cdot, \text{'sigmod'})] \text{ or } [contains(\cdot, \text{'vldb'})]]]]$

$L_5: //\text{inproceedings}[contains(\text{booktitle}, \text{'SIGMOD'})]$

$L_6: //\text{inproceedings}[contains(\text{title}, \text{'Data Mining'})]$

$L_7 = L_2 \cap L_3 \cap L_4 \cap L_5 \cap L_6: //\text{inproceedings}[P]$

# A Development of Hash-Lookup Trees to Support Querying Streaming XML

James Cheng and Wilfred Ng

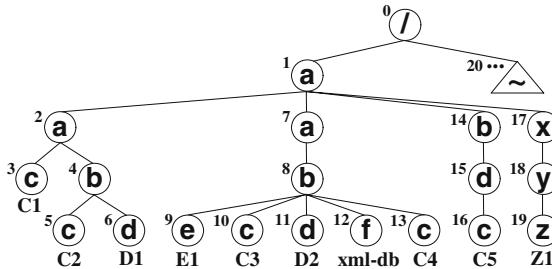
Department of Computer Science and Engineering,  
The Hong Kong University of Science and Technology, Hong Kong  
`{csjames, wilfred}@cse.ust.hk`

**Abstract.** The rapid growth in the amount of XML data and the development of publish-subscribe systems have led to great interest in processing streaming XML data. We propose the QstreamX system for querying streaming XML data using a novel structure, called Hash-Lookup Query Trees, which consists of a Filtering HashTable (FHT), a Static Query Tree (SQT) and a Dynamic Query Tree (DQT). The FHT is used to filter out irrelevant elements and provide direct access to relevant nodes in the SQT. The SQT is a tree model of the input query. Based on the SQT, the DQT is built dynamically at runtime to evaluate queries. We show, with experimental evidence, that QstreamX achieves throughput five times higher than the two most recently proposed stream querying systems, XSQ and XAOS, at much lower memory consumption.

## 1 Introduction

With the rapid growth in the amount of XML data, processing streaming XML data has gained increasing attention in recent years. Two main and closely related stream processing problems in XML are *filtering* [16, 5, 2, 7, 8, 13] and *querying* [3, 10, 11, 14]. The problem of filtering is to match a set of boolean path expressions (usually in XPath syntax) with a stream of XML documents and to return the identifiers of the matching documents or queries. In querying streaming XML data, however, we need to output all the elements in the stream that match the input query. Apart from natural streaming data used in publish-subscribe systems such as stock quotes and breaking news, it is sometimes more feasible to query large XML datasets in a streaming form, since we need to parse the document only once and keep only data that are relevant to the query evaluation in the memory.

In this paper, we focus on processing XPath queries with streaming XML data. Unlike filtering, querying outputs an element if it matches the input query. The difficulty is that in the streaming environment, we sometimes cannot determine whether an element is in the query result with the data received so far. However, we cannot simply discard the element as its inclusion in the query result may be verified with some element arriving in the future. Therefore, we need to buffer the potential query results. Proper buffer handling for querying XML streams, however, is rather complex, as illustrated by the following example.



**Fig. 1.** A Sample XML Document Tree

**Example 1.** Consider evaluating the query  $Q = “//a[./f]//b/c”$  on the XML document tree in Figure 1, assuming its elements come as a stream in ascending order of their (numerical) *ids* marked near the circle.

When the element  $c_5$  (i.e. the node with label “c” and  $id = 5$  on the left side of the tree) arrives, we have two node sequences,  $q_1 = \langle a_1, b_4, c_5 \rangle$  and  $q_2 = \langle a_2, b_4, c_5 \rangle$ , matching the main path of  $Q$ , i.e. “ $//a//b/c$ ”. However, we cannot output  $c_5$  at this stage, since the predicate, “[./f]”, of both  $a_1$  and  $a_2$  have not been satisfied. As this predicate may be satisfied with an  $f$  element that comes later, we must *buffer*  $c_5$  for both  $q_1$  and  $q_2$ ; but *only one copy* of  $c_5$  should be kept in memory as to avoid *duplicate buffering*.

When the end-tag of the element  $a_2$  arrives,  $a_2$  expires and so does the node sequence  $q_2$ . Since  $a_2$ ’s predicate is not satisfied, we need to *remove* the element  $c_5$  buffered for  $q_2$ . But  $c_5$  should not be deleted, since it is still being buffered for  $q_1$ , which may satisfy  $Q$  if there is an  $f$  element, descendant of  $a_1$ , coming in the stream. Similarly, we buffer  $c_{10}$  for the node sequences,  $q_3 = \langle a_1, b_8, c_{10} \rangle$  and  $q_4 = \langle a_7, b_8, c_{10} \rangle$ . Then when the start-tag of the element  $f_{12}$  arrives,  $q_1$ ,  $q_3$  and  $q_4$  satisfy  $Q$ . Hence, we need to immediately *flush* the element  $c_5$  buffered for  $q_1$  and the  $c_{10}$  buffered for  $q_3$  and  $q_4$ . However, we should *flush*  $c_{10}$  *only once*, though it is buffered for both  $q_3$  and  $q_4$ .

When  $c_{13}$  arrives, we should not buffer but *output*  $c_{13}$  immediately, since this time the node sequences,  $\langle a_1, b_8, c_{13} \rangle$  and  $\langle a_7, b_8, c_{13} \rangle$ , instantly satisfy  $Q$ . Again, we should output  $c_{13}$  *only once* for the two sequences.

Example 1 suggests some important issues in the query processing: (1) *buffering* of potential query results or *outputting* determined query results; (2) the decision of *flushing* or *removing* buffered data; and (3) *duplicate avoidance* in buffering, outputting, flushing and removing. Let us call all these issues collectively as *buffer handling* in our subsequent discussion.

Buffering comes only with the presence of predicates. The query in Example 1 contains only a single atomic predicate but the problem is already very complex. Another important issue is that a substantial amount of elements in a stream are usually irrelevant, however, no existing querying systems have considered filtering out these elements.

We propose the QstreamX system, which attempt to address the above-mentioned challenges with the use of a novel data structure, called *Hash-Lookup*

*Query Trees (HLQT).* HLQT consists of the following three components: a *Filtering HashTable (FHT)*, a *Static Query Tree (SQT)* and *Dynamic Query Tree (DQT)*. The FHT filters out irrelevant streaming elements and provides direct access to nodes in the SQT that are relevant for the processing of relevant elements. The SQT is a tree model of the input query, based on which the DQT is constructed dynamically at runtime to evaluate queries.

QstreamX has the following desirable features:

**Language Expressiveness.** QstreamX supports all XPath axes except the sideways axes (i.e. `preceding-sibling` and `following-sibling`). It also supports multiple and nested predicates with `and` and `or` operators, a common set of aggregations, and multiple queries and outputs.

**Processing Efficiency.** Our algorithm is able to achieve  $O(|D|)$  time complexity and  $O(|Q|)$  space complexity, where  $|D|$  is the size of the streaming data and  $|Q|$  is the size of the input query.

**Buffering Effectiveness.** QstreamX (1) buffers only those data that *must* be buffered for the correct evaluation of the query; (2) flushes or removes buffered data with no delay; and (3) avoids buffering and outputting any duplicate data.

**Effective Design.** HLQT makes the implementation of QstreamX straightforward. The FHT is realized as a simple array that stores distinct query elements and pointers to the SQT nodes. The SQT is translated directly from the input query by four simple transformation rules, while the DQT is constructed with correspondence to the structure of the SQT.

In the rest of the section, we discuss related work on stream processing. In Section 2, we present the XPath queries supported by QstreamX. We define Hash-Lookup Query Trees and present query evaluation in Sections 3. We evaluate QstreamX in Section 5 and conclude the paper in Section 6.

## 1.1 Related Work

A number of *filtering* systems [16, 5, 2, 7, 13, 8] have been proposed to process XPath filters on streaming XML documents. XFilter [1] converts queries into separate Deterministic Finite Automata (DFAs), while YFilter [6] eliminates redundant processing on common prefixes in the queries by a single Non-Deterministic Finite Automaton (NFA). XTrees [5] also supports shared processing of common subexpressions of the queries by a trie. The throughput of these systems decreases linearly with the number of queries. LazyDFA [2, 7] ensures a constant high throughput by lazily constructing a DFA for the entire workload of queries. However, LazyDFA may require excessive memory for XML data with complex structures. This problem is addressed in [13], which clusters the queries into  $n$  DFAs to reduce the number of DFA states and introduces a shared NFA state table to reduce the size of the NFA state table stored in each DFA state. The XPush machine [8] eliminates common predicates by translating the query workload into a deterministic pushdown automaton. Among these systems, only [13] and [8] support almost the same set of queries (except aggregations) as QstreamX. Although

we consider the same query language, filtering only outputs the identifier of matching documents or queries and does not require buffering of potential query result.

A closer match to QstreamX is the XAOS algorithm [3], which translates an XPath query into a tree and uses an extra graph to support the `parent` and `ancestor` axes by converting them into forward axes. The graph determines which set of elements (and with what depth) to look for in the incoming stream. The tree is used to maintain a structure to keep track of the matched elements. However, the query results are only determined at the `ROOT` of the structure, i.e., at the end of the stream, while HLQT outputs an element no later than when its inclusion in the query result is decided. Keeping the matched data until the end of a stream also does not scale, especially because streaming data is unbounded. Moreover, features such as aggregations, `or`-expressions and multiple queries are infeasible in XAOS's approach.

The filtering systems [2, 7, 13, 8] guarantee a constant high throughput using a hash algorithm to access directly relevant states for processing each element. However, direct access to relevant states or nodes using hash-lookup is considerably complicated by buffer handling in the querying problem. In fact, all existing querying systems need to search for matching transitions or relevant nodes for each (including irrelevant) streaming element. Our proposed HLQT adopts a hash-lookup strategy, which is natural to filter out irrelevant elements and provide direct access to nodes relevant for processing relevant elements.

## 2 QstreamX Query Expressions

We support a practical subset of XPath 2.0 queries with extended aggregations, whose Extended Backus-Naur Form (EBNF) is shown in Figure 2.

```

Q ::= /LP (/OE)?
LP ::= LS | LS/LP
LS ::= AX::(tag | *) P? | (@attribute | @*) CP?
AX ::= self | child | descendant | descendant-or-self | parent |
 ancestor | ancestor-or-self
P ::= [P (and | or) P] | [LP CP?]
CP ::= OP literal | [[. OP literal] (and | or) [. OP literal]]
OP ::= > | < | >= | <= | != | contains | starts-with
OE ::= text() | count() | sum() | avg() | max() | min()

```

**Fig. 2.** EBNF Grammar of QstreamX Queries

## 3 Hash-Lookup Query Trees

We now define the three components of *Hash-Lookup Query Trees (HLQT)*: the *Static Query Tree*, the *Dynamic Query Tree* and the *Filtering HashTable*.

**The Static Query Tree.** The *Static Query Tree (SQT)* is a tree model of the input query constructed by four transformation rules, as depicted in Figure 3.

where elements in dotted line are optional components. The transformation rules are derived directly from the EBNF of the language presented in Figure 2.

We now explain the four transformations that are used to construct the SQT.

**(a) LocationStep Transformation.** A location step is transformed into an *SQT node*, or a *snode* for short, which is a triplet,  $(axis, predicate, dlist)$ , where *axis* is the axis of the location step; *predicate*, if any, is handled by Predicate Transformation; and *dlist* is a list of DQT node pointers that provide direct access to the DQT nodes. A *dlist* is initially empty, since node pointers are added to the *dlist* at runtime during query evaluation.

**(b) LocationPath Transformation.** A location path is a sequence of one or more location steps. Therefore, LocationPath Transformation is just a sequence of one or more LocationStep Transformations, where a *snode* is connected to its parent by its *axis*.

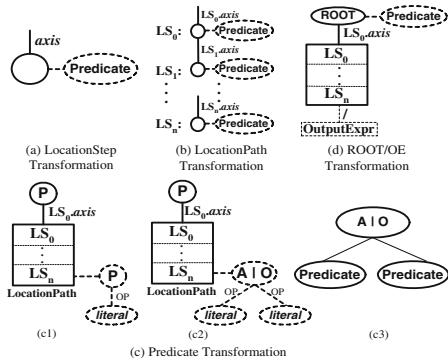


Fig. 3. SQT Transformation Rules

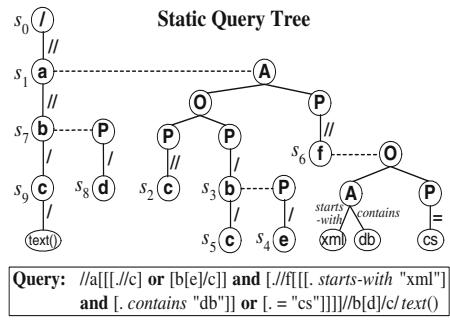


Fig. 4. The SQT of an Example Query

**(c) Predicate Transformation.** To facilitate efficient predicate processing, we require predicates be fully parenthesized when they are joined by the logical operators. We then model the predicates as a binary tree, called a *Predicate Binary Tree* (*PBT*). A node in the *PBT* is called an *SQT predicate node*, or a *spnode*, which is one of the following three types: *O* (for *or*-expression), *A* (for *and*-expression) and *P* (an encapsulation of other predicate). Value comparison, if any, is also modelled by a *P-spnode*, by an *A-spnode* or by an *O-spnode* for multiple value matches.

We classify predicate transformation into the following three categories: (1) an atomic predicate is transformed by applying LocationPath Transformation on the location path in the predicate, as shown in Figure 3(c1) and 3(c2); (2) a nested predicate is transformed by applying Predicate Transformation recursively; and (3) an *and/or* expression is transformed by applying Predicate Transformation on both sides of the logical operator, as shown in Figure 3(c3).

**(d) ROOT/OE Transformation.** This transformation is carried out in two steps. The first step is at the beginning of the SQT construction, we create the root of the SQT. The second step is at the completion of the SQT construction,

we create a node, called the *output node*, in order to model the output expression of the query.

Let  $s$  be a *snode*. If  $s$  has an ancestor that is a *spnode*, then we say  $s$  is *under* a PBT. Note that  $s$  is not part of the PBT, since a PBT consists of only *spnodes*. If the root of a PBT is connected to  $s$ , then the PBT is the PBT of  $s$ . We say that  $s$  is the *parent* of another *snode*,  $s'$ , if  $s$  and  $s'$  are connected by the *axis* of  $s'$ , and that  $s$  is the *indirect-parent* of  $s'$ , if  $s$  and  $s'$  are connected by a path of *spnodes* in the PBT of  $s$ .

The *primary path* of the SQT is the path that still remains when all PBTs and all *snodes* under the PBTs are removed. For example, in Figure 4, the nodes  $s_1$ ,  $s_3$ ,  $s_6$  and  $s_7$  have a PBT, while the nodes  $s_2$ ,  $s_3$ ,  $s_4$ ,  $s_5$ ,  $s_6$  and  $s_8$  are under a PBT;  $s_1$  is the parent of  $s_7$  but the indirect-parent of  $s_2$ ,  $s_3$  and  $s_6$ ;  $\langle s_0, s_1, s_7, s_9 \rangle$  is the primary path. Moreover, if a *snode* is not on the primary path, then it is under a PBT. Note that there may be more than one primary path in the DQT, if the streaming data is recursive with respect to an *axis* on the primary path of the SQT. The dot notation  $a.b$  means that  $b$  is the component of  $a$ . For example,  $s.dlist$  refers to the *dlist* of  $s$ .

**The Dynamic Query Tree.** The *Dynamic Query Tree (DQT)* is constructed dynamically at runtime to simulate the execution of query evaluation. We use the SQT to guide the construction of the DQT and to provide direct access (using the *dlists*) to nodes in the DQT that are relevant for the processing of a streaming element. We now detail the structure of the DQT, with reference to the SQT.

Like the SQT, there are two types of nodes in the DQT: *DQT node* (*dnode*) and *DQT predicate node* (*dpmode*). Each *dnode* (*dpmode*) corresponds to a unique *snode* (*spnode*) and the relationship between the *dnodes* (*dpmodes*) is the same as that between the corresponding *snodes* (*spnodes*).

A *dnode*,  $d$ , is a triplet,  $(\text{depth}, \text{blist}, \text{flag})$ , where *depth* is the depth of the corresponding XML element in the streaming document, and the *blist* and the *flag* are used to aid buffer handling and predicate evaluation. The content of  $d.blist$  is described as follows:

- If  $d$  is on the primary path, then  $d.blist$  is either  $\emptyset$  or a list of pointers to where query results are buffered.
- If  $d$  is under a PBT, then  $d$  is used to evaluate a predicate and hence no data need be buffered for  $d$ . However, we assign a special value,  $\rho$ , to  $d.blist$  so that we can immediately identify whether a *dnode* is under a PBT or on the primary path during query processing.

The *flag* is either T or F, which has different meanings:

- If  $d$  is on the primary path (i.e.  $d.blist \neq \rho$ ), :
  - Case of  $d.flag = T$ . The predicates of all  $d$ 's ancestors and  $d$  are satisfied.
  - Case of  $d.flag = F$ . The predicate of some of  $d$ 's ancestors has not been satisfied.

- If  $d$  is under a PBT (i.e.  $d.blist = \rho$ ):  
 Case of  $d.flag = T$ . All  $d$ 's descendants are satisfied.  
 Case of  $d.flag = F$ .  $d$  has some descendant not satisfied.

When we say that a *dnode*,  $d$ , is satisfied, we mean that the predicates of all  $d$ 's descendants and  $d$  are satisfied. When we say that  $d$ 's predicate is satisfied, we mean that  $d$ 's PBT is evaluated to be true (and deleted), but it does not imply that the predicates of  $d$ 's descendants are all satisfied. A *dnode* is one of the following types: P, A (i.e. `and`), O (i.e. `or`), L (i.e. left) and R (i.e. right), where L (or R) indicates that the left (or right) side of the `and`-predicate has been satisfied and only the right (or left) side needs to be processed.

**The Filtering Hashtable.** The Filtering HashTable (FHT) filters out all streaming elements that do not match any element in the query. A hash value is generated for each distinct element or attribute label in the query. The labels are then stored in the corresponding hash slot. Collision is handled by chaining. In practice, collisions are very rare in QstreamX, since we use a hashtable of default size 1024 (only a few KB memory size), while most XML datasets have less than 200 distinct elements.

To provide direct access to *snodes* that match a streaming element, a list, called the *slist*, is kept in each hash slot. An element of the *slist* is a triplet,  $(sparent, schild, dp)$ , where *sparent* and *schild* are two *snodes* pointers, and *sparent* is either the parent or indirect-parent of *schild*; and *dp* is a list of L or R symbols to represent the left or right direction, respectively, from *sparent* to *schild*, if *sparent* is the indirect parent of *schild* and *sparent*'s PBT has more than one *snodes*; *dp* is denoted by  $\emptyset$  otherwise.

Figure 5 shows the *slist* of the six elements, **a**, **b**, **c**, **d**, **e** and **f**, of the query in Figure 4. For example, **b**'s *slist* has two elements since there are two **bs** in the query. In both *slist*-elements, the *schilds*,  $s_7$  and  $s_3$ , model **b**; while the *sparent*,  $s_1$ , is the parent of  $s_7$  but the indirect-parent of  $s_3$ . The first *dp* is  $\emptyset$  since we can reach  $s_7$  from  $s_1$  directly, while the second *dp*, LR, shows that from the root of  $s_1$ 's PBT, we reach  $s_3$ 's parent by going left and then right.

```
a:{(s0,s1,())}; d:{(s7,s8,())}; e:{(s3,s4,())}; f:{(s1,s6,R)};
b:{(s1,s7,()),(s1,s3,LR)}; c:{(s7,s9,()),(s3,s5,()),(s1,s2,LL)}.
```

**Fig. 5.** The *slist* of the Query in Figure 4

## 4 QstreamX Query Processing

Consider the query shown in Figure 4 on the XML document presented in Figure 1. For brevity, we use  $1_i.S$  to denote the *S* event (same for *A*, *T* and *E*) of the element, whose label is *1* and id is *i*, in Figure 1. For example,  $a_1.S$  refers to the *S* event of  $a_1$ . Throughout, we use  $s_i$  to denote a *snodes* in the SQT (see Figure 4) and  $d_i$  to denote a *dnode* in the DQTs (see Figures 6(a)-6(f)).

**(a) Basic DQT Construction.** We first create the root of the DQT,  $d_0 = (0, \emptyset, T)$ , and add  $d_0$ 's pointer to the  $dlist$  of the corresponding  $snode$ ,  $s_0$ . On the arrival of  $a_1.S$ , we apply hashing on the label,  $a$ , and access  $a$ 's  $slist$  (c.f. Figure 5),  $\{(s_0, s_1, \emptyset)\}$ , that is stored in  $a$ 's hash slot. We use  $s_0$ 's pointer in  $a$ 's  $slist$  to access  $s_0$  and then use  $d_0$ 's pointer in  $s_0.dlist$  to access  $d_0$ . From  $d_0$  we create its child,  $d_1 = (1, \emptyset, F)$ , to correspond to  $s_0$ 's child,  $s_1$ . We set  $d_1.blist$  to  $\emptyset$ , since  $s_1$  is on the primary path, and  $d_1.flag$  to  $F$ , since  $s_1$  has a PBT. We then construct the PBT for  $d_1$  according to the PBT of  $s_1$  and insert the pointer to  $d_1$  into  $s_1.dlist$ . In the same way, for the next (recursive) event  $a_2.S$ , we create another child,  $d_2$ , for  $d_0$ . In the following discussion, when we create a  $dnode$ , we also construct its PBT, if any; and after the  $dnode$  is created, its pointer is inserted into the  $dlist$  of its corresponding  $snode$  to provide direct access. We show the DQT constructed so far in Figure 6(a), in which we also show all the non-empty  $dlists$  of the  $snodes$ .

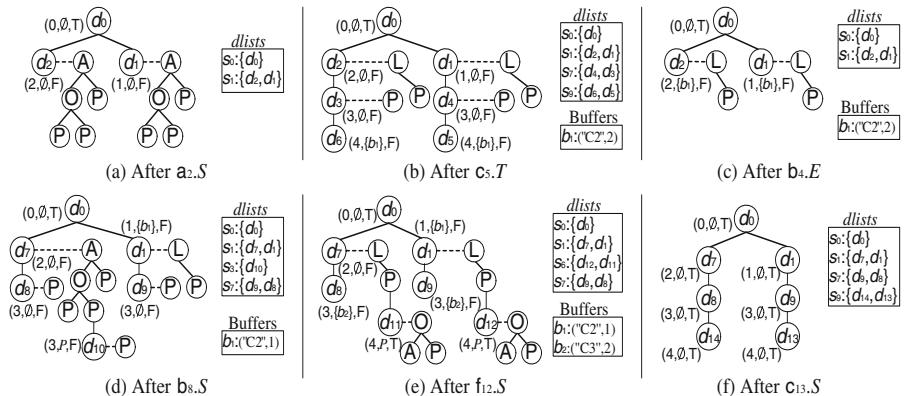


Fig. 6. The DQTs for Processing the Query in Figure 4 on the XML Doc in Figure 1

**(b) Predicate Processing (Bubble-Up).** The next streaming event is  $c_3.S$  and we have three elements in  $c$ 's  $slist$ :  $\{(s_7, s_9, \emptyset), (s_3, s_5, \emptyset), (s_1, s_2, LL)\}$ . However, the  $dlists$  of the parent  $snodes$ ,  $s_7$  and  $s_3$ , are empty, which implies that  $s_7$  and  $s_3$  have not been matched. Hence, we only process  $(s_1, s_2, LL)$  and access  $d_2$  and  $d_1$  via their pointers in  $s_1.dlist$ . We then use  $dp$ , i.e.  $LL$ , to start from the root of  $d_2$ 's PBT,  $p_r$ , to reach the leftmost leaf  $dnode$ ,  $p_l$ . Since  $s_2$  has no PBT and child,  $c_3.S$  satisfies  $s_2$ . Thus, no  $dnode$  is created but we bubble the satisfaction from  $p_l$  up the PBT. The bubble-up immediately satisfies  $p_l$ 's parent since it models an  $or$ -predicate. Hence, we continue bubbling up to  $p_r$ , which is an  $and$ -predicate. We change  $p_r.type$  to  $L$  to indicate that the left child of  $p_r$  is evaluated to be true. In the same way, we evaluate  $d_1$ 's PBT with  $c_3.S$ . We update the DQT in Figure 6(b). (We ignore  $d_3-d_6$  for the time being.)

**(c) Elimination of Redundant Processing.** We do not process  $c_3.T$  and  $c_3.E$ , since the  $dlists$  of  $s_9$ ,  $s_5$  and  $s_2$  are empty, implying that no  $dnode$  exists

to process  $c_3.T$  and  $c_3.E$ . Note that  $c_3.T$  and  $c_3.E$  are indeed redundant for processing the query .

Then it comes  $b_4.S$ . Using  $(s_1, s_7, \emptyset)$  in  $b$ 's *slist* we access  $s_1$  and then  $d_2$  and  $d_1$ . From  $d_2$  and  $d_1$  we create their respective child,  $d_3$  and  $d_4$ , corresponding to  $s_1$ 's child  $s_7$ . However, for the other element,  $(s_1, s_3, LR)$ , in  $b$ 's *slist*, when we use  $dp$  to process  $s_3$ , we find that  $s_3$  belongs to the satisfied part of a PBT, since the first component of  $dp$ , i.e. L, matches the type of the root of both  $d_2$ 's PBT and  $d_1$ 's PBT. This is also a part of QstreamX's mechanism to eliminate redundant processing. In the same way, we also skip the processing of last two *slist*-elements in  $c$ 's *slist* for the next streaming element,  $c_5$ .

**(d) Buffering.** We only need to process the *slist*-element,  $(s_7, s_9, \emptyset)$ , for  $c_5$ . For  $c_5.S$ , we access  $d_4$  and  $d_3$  via  $s_7.dlist$ , and create their respective child,  $d_5$  and  $d_6$ , corresponding to  $s_9$ . For  $c_5.T$ , we apply hashing on the label, c, obtained from the stack top. We then access  $d_6$  and  $d_5$  via  $s_9.dlist$ . Since  $s_9$ 's child is the output node and both  $d_6$  and  $d_5$  have no PBT,  $c_5.T$  is a potential query result. We create Buffer  $b_1$  to buffer  $c_5.T$ , i.e. "C2". Then we insert the pointer to  $b_1$  into both  $d_6.blist$  and  $d_5.blist$ , and increment  $b_1.counter$  twice. We show the updated DQT and the Buffer in Figure 6(b).

**(e) Uploading.** To process  $c_5.E$ , we use  $(s_7, s_9, \emptyset)$  to access  $s_9$  and then access  $d_6$  and  $d_5$ , via  $s_9.dlist$ . We upload  $d_6.blist$  and  $d_5.blist$  to their parents  $d_3$  and  $d_4$  respectively. Then, we delete  $d_6$  and  $d_5$ , and remove their pointers from  $s_9.dlist$ .

With  $d_6.S$  and  $d$ 's *slist*,  $\{(s_7, s_8, \emptyset)\}$ , we then further delete the PBT of  $d_4$  and  $d_3$ , since  $d_6.S$  satisfies  $s_8$ . Again,  $s_8$ 's empty *dlist* avoids the redundant processing of  $d_6.T$  and  $d_6.E$ .

To process  $b_4.E$ , we upload  $d_4.blist$  and  $d_3.blist$  to their parents  $d_1$  and  $d_2$  respectively. We then delete  $d_4$  and  $d_3$ , and remove their pointers from  $s_7.dlist$ . We update the DQT and the *dlists* in Figure 6(c). Note that both  $d_1.blist$  and  $d_2.blist$  now contain the pointer to Buffer  $b_1$ .

**(f) Buffer Removing.** Then for  $a_2.E$ , we access  $d_2$  and  $d_1$  via  $s_1.dlist$ . We do not upload  $d_2.blist$  since  $d_2$  has a PBT, i.e. the predicate is not satisfied, and hence the data buffered is not a query result with respect to  $d_2$ . We access Buffer  $b_1$  via  $b_1$ 's pointer in  $d_2.blist$  to decrement  $b_1.counter$ . Then we delete  $d_2$  and its PBT. We do not process  $d_1$ , since  $d_1.depth$  does not match the depth of  $a_2.E$ .

We then create another child,  $d_7$ , for  $d_0$  with  $a_7.S$ . Then corresponding to  $s_7$ ,  $b_8.S$  creates  $d_8$  and  $d_9$  as child of  $d_7$  and  $d_1$  respectively. Although  $b_8.S$  is not processed for  $d_1$ 's PBT, we create  $d_{10}$  to evaluate  $s_3$ , as shown in Figure 6(d).

Then  $e_9.S$  satisfies  $s_4$  and we delete  $d_{10}$ 's PBT, while  $s_4$ 's empty *dlist* avoids  $e_9.T$  and  $e_9.E$  being redundantly processed. Next  $c_{10}.S$  creates a child for  $d_9$  and  $d_8$  respectively, corresponding to  $s_9$ . This  $c_{10}.S$  also satisfies  $s_5$ , and the satisfaction triggers  $d_{10}$ 's satisfaction, which is bubbled up until it updates the type of the root of  $d_7$ 's PBT to L. The last element in  $c$ 's *slist* is thus not processed, since  $s_2$  belongs to a satisfied part of the PBT.

For  $c_{10}.T$ , i.e. "C3", we buffer "C3" in Buffer  $b_2$ . On the arrival of  $c_{10}.E$ , the *blists* are uploaded to  $d_9$  and  $d_8$ . Then  $d_{11}.S$  satisfies  $s_8$  and we delete the PBT

of both  $d_9$  and  $d_8$ . Next,  $f_{12}.S$  creates  $d_{11}$  and  $d_{12}$  to evaluate  $s_6$ , as shown in the updated DQT in Figure 6(e).

**(g) Predicate Processing (Trickle-Down) and Flushing.** Then  $f_{12}.T$ , i.e. “xml-db”, matches the `and`-predicate in  $d_{12}$ ’s and  $d_{11}$ ’s PBT. We bubble up the satisfaction to the `or`-predicate, i.e. the root of  $d_{12}$ ’s and  $d_{11}$ ’s PBT. Thus, both  $d_{12}$  and  $d_{11}$  are satisfied; and the satisfaction is bubbled up and triggers the satisfaction of both  $d_1$ ’s PBT and  $d_7$ ’s PBT. Since  $d_1$  and  $d_7$  are on the primary path, we trickle down the satisfaction of their PBT to their descendants.

The trickle-down starts at  $d_1$ , since  $d_{12}$ , which is under  $d_1$ ’s PBT, is processed before  $d_{11}$ . We first update  $d_1.flag$  to T and access  $b_1$  via  $d_1.blist$  to flush  $b_1$ . We then decrement  $b_1.counter$  to zero and hence we delete  $b_1$ . We also set  $d_1.blist$  to  $\emptyset$ . Then we trickle down to  $d_1$ ’s child  $d_9$ , we set  $d_9.flag$  to T and access  $b_2$  via  $d_9.blist$  to flush  $b_2$ . We then set  $b_2.store$  to “flushed” and decrement  $b_2.counter$ . Then we set  $d_9.blist$  to  $\emptyset$ . When the trickle-down reaches  $d_8$ , we access  $b_2$  again via  $d_8.blist$ . Since  $b_2.store$  is “flushed”, we only decrement  $b_2.counter$ . We delete  $b_2$  since  $b_2.counter$  now becomes zero.

**(h) Outputting.** Then for  $c_{13}.S$  we create  $d_{13}$  and  $d_{14}$  as child of  $d_9$  and  $d_8$  respectively, as updated in Figure 6(f). Since  $d_9.flag$  and  $d_8.flag$  are T,  $d_{13}.flag$  and  $d_{14}.flag$  are also set to T. Therefore, when we process  $c_{13}.T$  for  $d_{14}$ , we immediately output  $c_{13}.T$  as a query result. We also set a flag to indicate that  $c_{13}.T$  is outputted, so that we do not output it again when we process  $d_{13}$  next. The flag is then unset.

Then for  $c_{13}.E$ , we delete  $d_{14}$  and  $d_{13}$ ; for  $b_8.E$ , we delete  $d_9$  and  $d_8$ ; for  $a_7.E$ , we delete  $d_7$ .

**(i) Depth Mismatch and Hash-Lookup Filtering.** Although  $s_7$  is satisfied again with  $b_{14}$  and  $d_{15}$ ,  $c_{16}$  does not match the *depth* of the child of  $s_7$  and is hence filtered out. The elements,  $x_{17}$ ,  $y_{18}$  and  $z_{19}$ , have no corresponding hash slots and are hence filtered out. Finally, we delete  $d_1$  when  $a_1.E$  comes, while we delete  $d_0$ , i.e. the root of the DQT, to terminate the query processing at the end of the stream.

## 5 Experimental Evaluation

We evaluate QstreamX on two important metrics for XML stream processing: the *throughput* and the *memory consumption*. We compare its performance with two most recently proposed querying systems, the XSQ system V1.0 [14] and the XAOS system [3]. We use the following four real datasets [12]: the Shakespeare play collection (Shake), NASA ADC XML repository (NASA), DBLP, and the Protein Sequence Database (PSD). We ran all the experiments on a Windows XP machine with a Pentium 4, 2.53 GHz processor and 1 GB main memory.

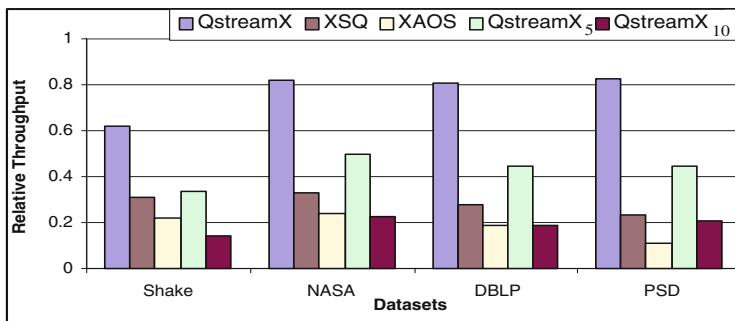
**Throughput.** Throughput measures the amount of data processed per second when running a query on a dataset. For each of the four real datasets, we use 10 queries, which have a roughly equal distribution of the four types:  $Q_1$  consists of only `child` axis,  $Q_2$  consists of only `descendant-or-self`;  $Q_3$  and  $Q_4$  mix the

two axes, but  $Q_3$  consists of a single atomic predicate, while  $Q_4$  allows multiple (atomic) predicates. An example of each type is shown below:

- ( $Q_1$ ): “/PLAY/ACT/SCENE/SPEECH/SPEAKER/text()”
- ( $Q_2$ ): “//dataset//author//lastname/text()”
- ( $Q_3$ ): “//inproceedings[year > 2000]/title/text()”
- ( $Q_4$ ): “//ProteinEntry[summary]/reference[accinfo]  
/refinfo[@refid = “A70500”]//author/text()”

The throughput<sup>1</sup> of each system on processing a single query is measured as the average of the throughput of processing each of the 10 queries for each dataset. We also measure the throughput of processing multiple queries (5 and 10 queries) by QstreamX, where the input queries are simply each half of the 10 queries and the 10 queries as a whole respectively. However, the Xerces 1.0 Java parser used in XSQ is on average two times slower than the C++ parser used in QstreamX and XAOS. Therefore, we use the *relative throughput* [14], which is calculated as the ratio of the throughput of each system to that of the corresponding SAX parser, to give a comparison only on the efficiency of the underlying querying algorithm.

As shown in Figure 7, QstreamX achieves very impressive throughput, which is about 80% of that of the SAX parser (the throughput for Shake is 78% when the dataset is scaled up by three time); in another word, 80% of the upper bound. Compared with XSQ and XAOS, QstreamX on average achieves relative throughput of 2.7 and 4.5 times higher, respectively. The tremendous improvement made by our algorithm over the XSQ and XAOS algorithms is mainly due to the effective filtering of irrelevant elements by hash-lookup and the direct access to relevant nodes through *slist* and *dlist*. Finally, we remark that the raw throughput of QstreamX is on average 5.4 and 9 times higher than that of XSQ and XAOS, respectively.



**Fig. 7.** Relative Throughput

The average relative throughputs of QstreamX on processing 5 queries and 10 queries are 43% and 19%, as denoted by QstreamX<sub>5</sub> and QstreamX<sub>10</sub> respectively in Figure 7. The great drop in the throughput is mainly because 5 and

<sup>1</sup> Since outputting the query results to the screen dominates the processing time, we write the results to a disk file for all systems.

10 times more potential query results need to be processed and duplicate avoidance has to be performed for 5 and 10 more times. However, this overhead is inevitable for processing multiple queries on XML streams, since we must buffer the potential query results at any given time. Despite of this, we remark that the throughput of QstreamX on 5 queries is still 1.5 times higher than that of XSQ (i.e. a raw throughput of 3 times higher), while that on 10 queries is only slightly lower (but a slightly higher raw throughput).

**Memory Consumption.** We measured roughly constant memory consumption of no more than 1 MB for QstreamX on all datasets and queries (including the two cases of multiple query processing). In fact, a large portion of the memory is used in buffering and in the input buffer of the parser, while the memory used for building the trees is almost negligible. The constant memory consumption proves the effectiveness of buffer handling, while the lower memory consumption verifies that the size of the DQT is extremely small. The memory consumption of XSQ is also constant (as a result of its effective buffering) but several times higher than that of QstreamX (as a result of a less efficient data structure). The memory consumption of the XAOS system increases linearly, since the algorithm stores both the data and the structure of all matched elements and outputs the results at the end of a stream.

## 6 Conclusions

We have presented the main ideas in QstreamX, an efficient system for processing XPath queries of streaming XML data, by utilizing a novel data structure, Hash-Lookup Query Trees (HLQTs), which consists of a simple hash table (the FHT) and two elegant tree structures of the SQT and the DQT. We have devised a set of well-defined transformation rules to transform a query into its SQT and discussed in detail how the dynamic construction of the DQT evaluates queries. A unique feature of QstreamX is that it processes only relevant XML elements in the stream by hash-lookup and accesses directly nodes that are relevant for their processing. We have demonstrated that QstreamX achieves significantly higher throughput and consumes substantially lower memory than the state-of-the art systems, XSQ and XAOS.

## References

1. M. Altinel and M. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *Proceedings of VLDB*, 2000.
2. I. Avila-Campillo and et al. XMLTK: An XML Toolkit for Scalable XML Stream Processing. In *Proc. of PLANX*, 2002.
3. C. Barton and et al. Streaming XPath Processing with Forward and Backward Axes. In *Proceedings of ICDE*, 2003.
4. Z. Bar-Yossef, M. F. Fontoura, and V. Josifovski. On the Memory Requirements of XPath Evaluation over XML Streams. In *Proceedings of PODS*, 2004.
5. C. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient Filtering of XML Documents with XPath Expressions. In *Proceedings of ICDE*, 2002.

6. Y. Diao, P. Fischer, M. Franklin, and R. To. YFilter: Efficient and Scalable Filtering of XML Documents. In *ICDE*, 2002.
7. T. Green, G. Miklau, M. Onizuka, and D. Suciu. Processing XML Streams with Deterministic Automata. In *Proceedings of ICDT*, 2003.
8. A. Gupta and D. Suciu. Stream Processing of XPath Queries with Predicates. In *Proceedings of SIGMOD*, 2003.
9. V. Josifovski, M. F. Fontoura, and A. Barta. Querying XML Streams. In *VLDB Journal*, 2004.
10. M. L. Lee, B. C. Chua, W. Hsu, and K. L. Tan. Efficient Evaluation of Multiple Queries on Streaming XML Data. In *Proceedings of CIKM*, 2002.
11. B. Ludascher, P. Mukhopadhyay, and Y. Papakonstantinou. A Transducer-Based XML Query Processor. In *Proceedings of VLDB*, 2002.
12. G. Miklau and D. Suciu. XML Data Repository. <http://www.cs.washington.edu/research/xmldatasets>.
13. M. Onizuka. Light-weight XPath Processing of XML Stream with Deterministic Automata. In *Proceedings of CIKM*, 2003.
14. F. Peng and S. Chawathe. XPath Queries on Streaming Data. In *Proceedings of SIGMOD*, 2003.

# Efficient Integration of Structure Indexes of XML

Taro L. Saito and Shinichi Morishita

University of Tokyo, Japan  
`{leo, moris}@cb.k.u-tokyo.ac.jp`

**Abstract.** Several indexing methods have been proposed to encode tree structures and path structures of XML, which are generally called *structure indexes*. To efficiently evaluate XML queries, it is indispensable to integrate tree structure and path structure indexes as a multidimensional index. Previous work of XML indexing have often developed specialized data structures tailored to some query patterns to handle this multidimensionality, however, their availability to the other types of queries has been obscure. Our method is based on the multidimensional index implemented on top of the B+-tree, and also it is general and applicable to the various choice of XML labeling methods. Our extensive experimental results confirm great advantages of our method.

## 1 Introduction

XML databases require the capability to retrieve nodes by using a variety of structural properties of XML, which are basically derived from *tree structures* of XML such as document order of nodes, subtree, sibling, ancestor, descendant nodes, etc. The other properties are *path structures* of XML, that consist of sequences of tag and attribute names, e.g. `//news/Japan`. These various aspects of XML make its query processing difficult, and index structures for this purpose, which are generally called the *structure indexes* [1], have attracted research attention. Most of the proposed structure indexes aim to efficiently process XPath [2] queries, which is the *de facto* standard for navigating XML. XPath contains a mixture of tree and path traversal with several axis steps, e.g. the child-axis (`/`), the descendant-axis (`//`), ancestor-axis, sibling-axis, etc.

Since 1996, as XML gradually has established its position as a data representation format, tremendous number of structure indexes have been proposed, which are optimized for specific query patterns, including structural joins [3][4], twig queries [5], suffix paths [6], ancestor queries [7], etc. They are proved to be fast for their targeted queries, however, most of them introduce special purpose data structures implemented on disks, and ends up losing *flexibility* of choices of node labels. For example, XR-tree [7], which is optimized for retrieving ancestor nodes that have specific tag names, cannot incorporate other efficient path labels such as *p-labels* [6], which is the fastest for suffix-path queries. That means XR-tree achieves fast ancestor query performance in exchange for the performance of suffix path queries.

Care should be taken to devise a specialized data structure on a disk, since an industrial strength DBMS has to support transaction management, but its implementation cannot be dependent from several essential components of the DBMS; page buffer, lock manager, database logging for recovery, and also access methods, such as B+-trees or

R-trees [8]. These modules seem to be able to implement independently, however, all of them have a lot of interdependencies. Index structures of DBMS usually include intricate protocols for latching, locking, and logging. The B+-trees in serious DBMSs are riddled with calls to the concurrency and recovery code, and this logic is not generic to all access methods [9]. That is a reason why the transaction management of R-tree, which is famous as a multidimensional index structure, is not seriously supported in most of the DBMS products, including both of commercial and open-source programs.

A natural question that follows is whether we can utilize a B+-tree, which is a well-established disk-based data structure, to achieve good performance for various types of XML queries. Our answer to this question is affirmative. In this paper, we show XPath queries can be performed with combinations of only two types of indexes; tree-structure and path-structure indexes. A challenging problem is that these scans must be performed in a combined way, for example, we have to query ancestor nodes that belong to some suffix path.

Our approach to this problem is to integrate tree-structure and path-structure indexes into a multidimensional index implemented on a B+-tree. It accelerates query processing for complex combinations of structural properties. And also, it is possible to incorporate various types of labeling methods. As an integration approach, constructing multiple secondary B+-tree indexes does not help multidimensional query processing, since they work for only a single dimension, not the combinations of multiple dimensions. Moreover, the existence of multiple secondary indexes not only enlarges the database size, but also deteriorates the update performance. We overcome these obstacles by using space-filling curve technique [10][11][12] to align XML nodes in a multidimensional space into one-dimensional space so that these nodes can be stored in a single B+-tree. We show this approach is beneficial in both of the query performance and database size.

There are hundreds of combinations of labeling strategies for XML and some of them demand special purpose data structures implemented on disks. What we would like to reveal in this paper is how the *integration* of tree and path structure indexes works for various types of queries consisting of combinations of structural properties.

Our major contributions in this paper are as follows:

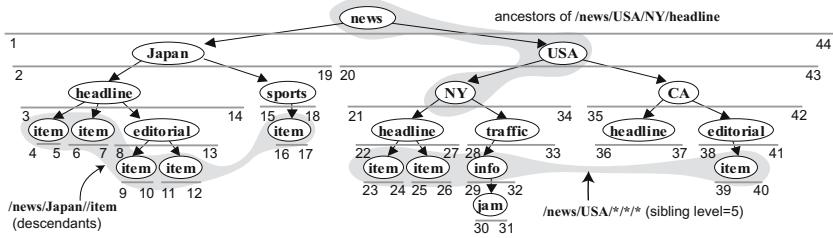
- We introduce an efficient multidimensional index structure, which is a combination of existing node labeling strategies in literature. While some XML indexes facilitate a few set of query patterns, our index is adaptive for various types of queries.
- We show an implementation of the proposed multidimensional index on top of the B+-tree, utilizing the space-filling curve technique.
- We show the multidimensional range query algorithm that can be performed without changing the B+-tree implementation.

Based on the above techniques, we have implemented an XML database system called **Xerial** (pronounced as [eksiriəl])<sup>1</sup>. Our experiments in Section 4 demonstrate Xerial's all-around performance for various types of queries. In spite of this faculty, its index size remains compact.

Organization of the rest of the paper is as follows: in Section 2, we explain tree-structure and path-structure indexes of XML, and show examples that motivate the

---

<sup>1</sup> Our system will be available at <http://www.xerial.org/>



**Fig. 1.** Interval labels of XML

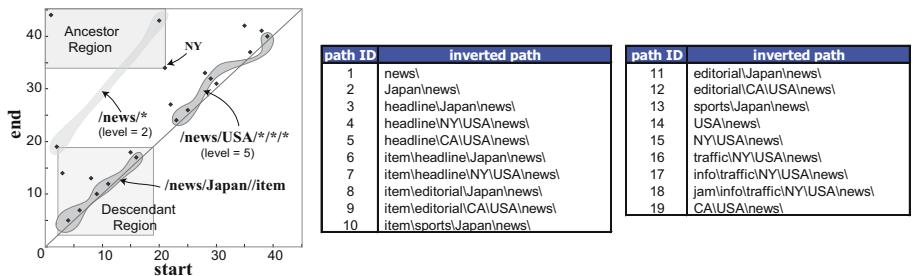
need of multidimensional index for XML In Section 3, we introduces its design and implementation. In Section 4, we provide results of experimental evaluation. Finally, we report related work in Section 5 and conclude in Section 6.

## 2 Backgrounds

**Tree-Structure Indexes.** XML has a tree structure, however, ancestor and descendant axes cannot be efficiently evaluated with standard tree navigations such as depth-first or breadth-first traversals. To make faster the process of ancestor-descendant queries, two types of node labeling methods have been developed; the *interval label* [3] and the *prefix label* [13]. The interval label (see also Fig. 1) utilizes containment of intervals to represent ancestor-descendant relationships of XML nodes. For example, if an interval label of a node  $p$  contains another interval of a node  $q$ ,  $p$  is an ancestor of  $q$ . The prefix label assigns node id paths from the root node to each node so that if the label of a node  $p$  is a prefix of the label of a node  $q$ ,  $p$  is an ancestor of  $q$ . Both of the node labeling strategies are fundamentally same in that they are designed to instantly detect ancestor-descendant relationships of two XML nodes. This operation is called *structural joins* [14]. Fig. 1 shows an example of the interval label. The interval assigned to each node subsumes intervals of its child and descendant nodes. These node labels are favorable in that they can be aligned in the document order of nodes by seeing start values of these intervals. Therefore, these labels can also be used to traverse the tree structures of XML in both of the depth-first and breadth-first manner. We call this type of indexes for tree navigation, *tree-structure* indexes.

**Path-Structure Indexes.** To efficiently evaluate path expression queries, in addition to the tree-structure indexes, we also need the *path-structure* indexes, which reduce the overhead of tree navigation by clustering nodes that belong to the same tag or attribute name paths. There are many proposals how to encode path structures of XML, varying from simply assigning an integer id to each independent path in the XML document [1] to creating a summary graph of path structures [15,16].

When a query contains the descendant-axis (//), we can localize the search spaces for the descendant nodes. For example, an XPath query //Japan/item can be decomposed into two paths; //Japan and //item, and the search space for //item will be localized



**Fig. 2.** Projection of the interval labels on a 2D-plain (left). Inverted path labels (right).

according to the results of //Japan (Fig. 1 and Fig. 2). Therefore, the tree-structure and path-structure indexes should be integrated to evaluate these types of queries.

In addition, we usually have to query not only with tag names of XML nodes but also with *suffix paths*. For example, an XPath //Japan//headline/item contains a suffix path //headline/item. Rather than querying headline and item nodes individually, it is far more efficient to scan nodes that have suffix paths //headline/item directly, since there are many item nodes whose parents are not the headline nodes. To improve accessibility to suffix paths, the p-label has been proposed [6]. The essence of its technique is to invert the sequences of paths occurring in the XML document, which we call *inverted paths*, and align these inverted paths in the lexicographical order considering each tag or attribute name in the paths as a comparison unit. Fig. 2 shows an example of inverted paths, where each inverted path is labeled with an integer id. To evaluate an XPath query //item, we have to collect nodes whose inverted path ids are contained in the range [6, 11). When a more detailed path is specified, for example, //headline/item, the query range narrows to [6, 8). With the inverted path ids, we can perform a suffix path query with a range search.

**Multidimensional Aspects of XML.** Here, we show why the integration of tree-structure and path-structure indexes is so important. Fig. 2 shows the mapping of the intervals (start, end) in Fig. 1 into a two-dimensional plane. A benefit of the interval labeling is that we can enclose all ancestor (descendant) nodes of some nodes within its upper left (lower right) rectangular region. For example, all ancestor nodes of the NY node (21, 34) is enclosed in its upper left rectangle. The process of a query, say /news/Japan//item, has to accurately extract item nodes within the subtree rooted by Japan (a shaded region in the figures). Since some item nodes exist out of this region, the index structure for XML demands the capability to capture nodes by the combination of start, end, and a path.

Although the 2D plane is useful to track ancestor and descendant nodes, we also need the information of the node depth (level) to process wild-cards in path expressions. For example, XPath expressions /news/\* and /news/US/\*/\*, which are useful to investigate the structure of XML database, require the level information to efficiently collect the answer nodes, since without indexes for the level values, its process has to traverse the tree-structure in depth-first or breadth-first manner; it is inefficient when the depth is

deep. Our experimental results confirm the inefficiency of these tree traversal methods for wild-card queries, i.e. the sibling-axis steps.

XPath [2] has 11 types of axis steps for tree navigations, that are *ancestor*, *descendant*, *parent*, *child*, *attribute*, *preceding-sibling*, *following-sibling*, *ancestor-or-self*, *descendant-or-self*, *preceding*, and *following*<sup>2</sup>. Among them, the six types of axis steps, *ancestor(-or-self)*, *descendant(-or-self)*, preceding and following, can be processed with the two-dimensional indexes for the interval labels (start, end). The *parent*, *child*, *preceding-sibling* and *following-sibling* axis-steps require all of start, end, level values, since start and end values are not sufficient to detect parent-child relationships of nodes. If attribute nodes of XML are modeled as child nodes of tags, the *attribute-axis* can be seen the same with the *child-axis*. Therefore, all of 11 axis-steps can be processed with the combination of (start, end, level) indexes, i.e. tree-structure indexes.

In addition to the tree-structure indexes, if we have the path-structure indexes, we can efficiently answer XPath queries, even if these answers are contained in meandering regions as illustrated in the query region for /news/Japan/item in Fig. 2. Therefore, multidimensionally indexing tree-structures and path-structures of XML is a key to accelerate XML query processing.

### 3 Multidimensional XML Index

In order to construct XML databases, we utilize a combination of tree-structure indexes, and path-structure indexes. Although, there are many proposals for labeling each type of index, we adopted labels that can be easily represented with integers.

We encode every XML node with a label:

(start, end, level, path, text),

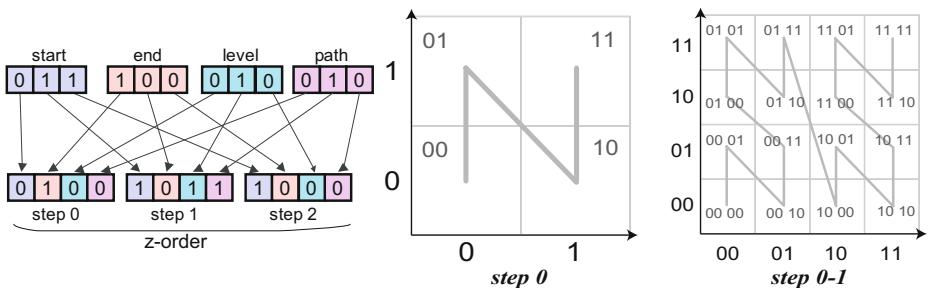
where start and end represent interval labels of XML, and level is the node depth. The path is the inverted path id described in Section 2. The text is a text content enclosed in the tag or attribute. Every attribute element in XML is assigned the same interval and level value with its belonging tag, so as to learn the subtree range of the tag from the index of the attribute node.

Although we utilized the interval labels for tree structures, other labeling schemes, such as prefix labels, can substitute them; the XML label will be (prefix-label, level, path, text). Each prefix label contains all prefix labels of its ancestor nodes, so there is no need to have end values for ancestor queries. The path labels also can be replaced simply with tag IDs or other labels.

The above labeling scheme is used to create multidimensional indexes. To index multidimensional data, it is general to use R-tree, which groups together nodes that are in close spatial proximity. However, implementations of the R-tree are not yet as matured as the B+-tree, which is broadly employed in the industrial strength DBMSs, while the R-tree is not. Although the B+-tree is a one-dimensional index structure, we can store the multidimensional data into a B+-tree by using a space-filling curve [12], such as

---

<sup>2</sup> The other two axis steps defined in XPath [2] are the *namespace* and *self* axis, which do not require any tree traversal.



**Fig. 3.** Interleave function generates a z-order from an index (start, end, level, path), that specifies a position on the z-curve

Hilbert curve, Peano curve etc. The space-filling curve traces the entire multidimensional space with a single stroke, and it can be used to align multidimensional points in one dimensional space.

However, what kind of space-filling curve is suited for XML indexing? To answer this question, let us confirm our objective to construct a multidimensional index; that is to make clusters of nodes that have same attribute values as possible, for example, same level values and same suffix paths, so that we can efficiently query nodes with combinations of these attribute values, i.e. start, end, level and path.

To meet this demand, we chose a straightforward approach; *bit-interleaving* of coordinate values. It gives a position on the *z-curve* [10][12], which is also a space-filling curve. The interleave function illustrated in Fig. 3 receives coordinate values of a point as input, and from their bit-string representations, it retrieves single bits from heads of coordinate values in a round-robin manner, then computes the *z-order*, which is an absolute position on the z-curve (Fig. 3). This linear ordering of XML nodes enables us to implement the multidimensional index on top of the B+-tree. In addition, each step in the z-order in Fig. 3 has a role to split each dimension. The first step splits each dimension into two, and the second step split each slice into 2, resulting in  $2^2 = 4$  slices, and so on. If two nodes are close in the multidimensional space, their z-orders also likely to be close in the some steps. It means these nodes will be probably placed in the same leaf page or its proximate pages in the B+-tree; this property is the nature of bit-interleaving.

**Normalizing Index Resolution.** The interleave function extracts bits beginning from the MSB (most significant bit). When value domains of the interleaved indexes are far different, for example, the domain of start values is  $0 \leq start < 2^{10}$ , and that of level values is  $0 \leq level < 2^3$ , the change of a value in a smaller domain has as equal significance to the z-order as that of the larger domains. In general, the depth of XML documents is not greater than 100, while the interval label for XML requires as large a value as the number of nodes, which can be more than 100,000. Thus, if we use the same bit-length number to represent each index value, the level values are less important in the z-order, and we fail to separate XML nodes level by level, deteriorating the sibling query performance.

To avoid this problem, we adjust the *resolution* of each index which is the maximum bit length that is enough to represent all values in the index domain. We denote the resolution of an index as  $r$ . For example, when a domain of some index is a range  $[0, v_{max}]$ , its resolution  $r$  is  $\lceil \log_2 v_{max} \rceil$ . The  $normalize_m(v)$  function converts an integer value  $v$ , whose resolution is  $r$ , into an  $m$ -bit integer value. We define  $normalize_m(v) = \lfloor v/2^{r-m} \rfloor$ , ignoring the fraction. For example, when  $m = 8$  and the resolution of each index of (start, end, level, path) is 10, 10, 3 and 4, respectively, an XML index (100, 105, 3, 2) is normalized to the 8-bit values (25, 26, 96, 32). By using normalized index values to compute z-orders, we can adjust the resolution so that level or path values, whose domains are usually small, affect much more to the z-order than start or end values. We simply denote this normalization process for some node  $p$  as  $normalize(p)$ .

**Range Query Algorithm.** The idea that utilizing z-curve for multidimensional indexes is first mentioned in the zkd-BTree [10] and is improved in the UB-tree [17]. Although both of them extended the standard B+-tree structure to make it efficient for multidimensional queries, we introduce a multidimensional range query processing algorithm without modifying existing B+-tree structures. In our algorithm, we need only two standard functions for the B+-tree; *find* and *next*. The *find( $k$ )* receives an key value  $k$  and finds the smallest entry whose key value is greater than or equal to  $k$ . The *next( $e$ )* returns the next entry of an entry  $e$  in the B+-tree.

We denote the z-order of a node  $p = (\text{start}, \text{end}, \text{level}, \text{path})$  as  $zorder(p)$ , and coordinates specified by the z-order  $z$  as  $coord(z)$ . Then,  $coord(zorder(p)) = p$ . Each entry in the B+-tree has the structure:  $zorder(normalize(p)) \Rightarrow p$ , where the left-hand side is a key to be used to sort XML nodes in the z-order. To perform a multidimensional query for a hyper-rectangle region  $Q(p_s, p_e)$ , where  $p_s$  and  $p_e$  are the multidimensional points specifying the beginning and end points of the query range, we can utilize a property of z-orders; all points  $p$  in the query range  $Q$  satisfies  $zorder(p_s) \leq zorder(p) \leq zorder(p_e)$  [17].

**Algorithm 1** shows the range query algorithm, and Fig. 4 illustrates its behavior. Since all nodes are aligned in the z-order in the B+-tree, we have to scan the key range of z-order from  $zorder(normalize(p_s))$  to  $zorder(normalize_{ceil}(p_e))$ , where  $normalize_{ceil}$  is calculated from  $\lceil v/2^{r-m} \rceil$  of each coordinate value  $v$ . That z-orders computed from normalized coordinate values may have round errors, so there is a case that  $coord(normalize(p))$  is contained in the normalized query range  $NQ(normalize(p_s), normalize_{ceil}(p_e))$ , but  $p$  is not in  $Q$ , since if we de-normalize  $NQ$ , illustrated in Fig. 4 as pseudo-query range, it is always equal to or larger than  $Q$ . Even though, the containment test for  $NQ$  (Step 10) is useful to detect whether the current z-order is completely out of range of  $Q$ . In this case, we can compute the *nextZorder* that re-enters into the query box  $NQ$  (Step 17). It skips some nodes in the outside of the query box and saves disk I/O costs. An efficient algorithm to compute next z-orders is described in [18]; this algorithm locates the most-significant bit-position, say  $j$ , in the z-order that can be safely set to one without jumping out of the query range, then adjusts other bit values which are lower than  $j$  so that the z-order becomes the smallest one contained in the query range but larger than the original z-order.

---

**Algorithm 1.** Range query algorithm

---

**Input:**  $Q(p_s, p_e)$  : query range  
**Output:** A node set within the query range

- 1:  $NQ = (\text{normalize}(p_s), \text{normalize}_{\text{ceil}}(p_e))$  // normalized query range of  $Q$
- 2:  $z_s = \text{zorder}(p_s), z_e = \text{zorder}(p_e)$
- 3:  $z = z_s$  // set the initial z-order to the beginning of the query range
- 4: // find an entry  $e$  in the B+-tree that has the smallest z-order larger than  $z$ .
- 5:  $e = \text{find}(z)$
- 6: **while**  $e$  is not *nil* **do**
- 7:    $z = e.z$  //  $e.z$  is the z-order (key value) of the entry  $e$
- 8:   **if**  $z > z_e$  **then**
- 9:     **return** // end of the query
- 10:   **if**  $\text{coord}(z)$  is contained in  $NQ$  **then**
- 11:     **while**  $e$  is not *nil* and  $e.z == z$  **do**
- 12:       // retrieve nodes whose z-order is  $z$  in the B+-tree
- 13:       **if** the entry  $e$  is contained in  $Q$  **then**
- 14:         output  $e$
- 15:        $e = \text{next}(e)$  // move to the next entry of  $e$  in the B+-tree
- 16:   **else**
- 17:      $\text{nextZorder} = \text{the smallest z-order larger than } z \text{ and contained in } NQ$ .
- 18:      $e = \text{find}(\text{nextZorder})$

---

## 4 Experimental Evaluation

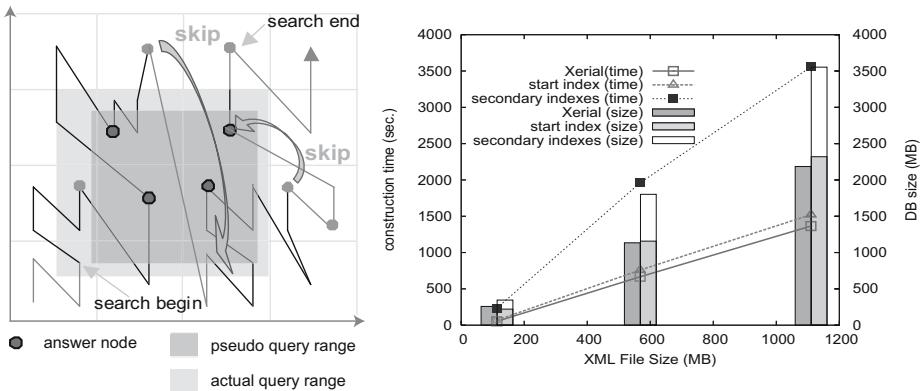
We evaluated the query performance of Xerial for several kinds of queries, e.g. ancestor, descendant, sibling, and path-suffix queries, which are the basic components to process more complicated queries such as structural joins, twig-queries, etc.

To clarify the benefit of our method, we prepared two competitors for Xerial; start index and path-start index. The start index simply sorts XML nodes in the order of start values. It has the data structure ( $\text{start} \Rightarrow \text{end, level, path, text}$ ) in B+-tree. The path-start index,  $((\text{path}, \text{start}) \Rightarrow \text{end, level, text})$ , sorts nodes first by path, then by start orders. These structure can localize search space of path queries within some subtree range, and similar structure is utilized in [4]. However, the following experiments reveal that such simple integration of indexes has several weak points.

**Implementation.** All of the indexes are implemented in C++. Xerial's index structure is  $\text{z-order} \Rightarrow (\text{start, end, level, path, text})$ . Every z-order is represented with 64-bit integer, and it is a sort key in the B+-tree. And also, all indexes hold start, end, level and path values as 32-bit integers. To construct B+-trees, we used the BerkeleyDB library [19], and their page sizes are set to 1K.

**Machine Environment.** As a test vehicle, we used an Windows XP, Pentium M 2GHz notebook with 1GB main memory and 5,400 rpm HDD (100GB).

**Database Size.** We compared database sizes of start index and Xerial. Fig. 4 shows their actual database sizes and construction times for various scaling factors (1 to 10) of the XMark's benchmark XML documents [20]. The secondary index in Fig. 4 shows the database size if we constructs three B+-tree indexes for end, level, path values to complement the functionality of the start index. Even though Xerial has additional z-orders, its database size is almost the same with the start index, and also it is much more compact than creating multiple secondary indexes. It is mainly because the B+-tree of Xerial has many duplicate entries having same z-orders, and it makes lower the depth of the B+-tree.



**Fig. 4.** Range query algorithm (left). DB construction time and DB size (right)

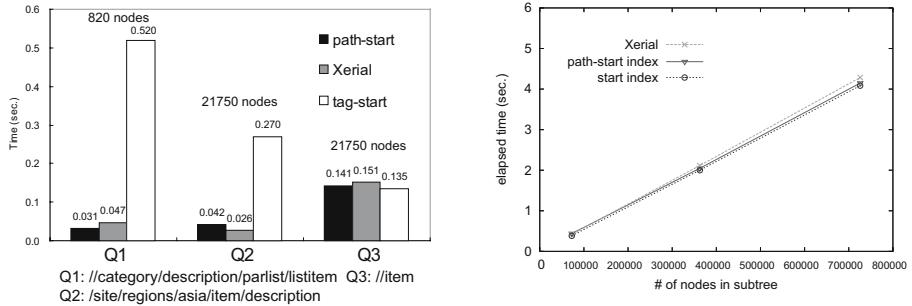
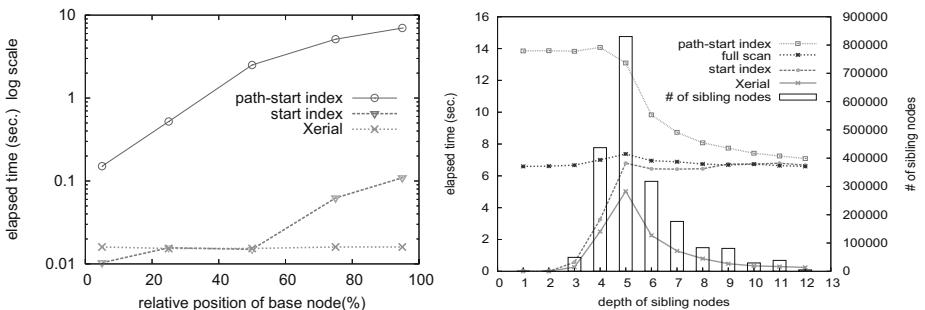
The following experiments are conducted on a XMark document (113MB, scaling factor = 1.0), and we measured the average times for individual query operations, ignoring the output costs of reporting the query results.

**Suffix Path Query.** First, we compared performance of suffix path queries. Fig. 5 shows how fast each index can collect nodes that have the same path suffixes. The path-start index, which has clusters of suffix paths is the fastest, and Xerial performs as fast as the path-start index, because the interleave function of Xerial also plays a role to group together nodes which have the same suffix path. The start index is weak in processing this kind of query since it has to scan the whole index, since information of path is hidden in its data pages.

In order to show that the importance of having flexibility for the choice of node labels, we also compared the performance of suffix path queries when inverted path cannot be used. The tag-start index uses tag IDs instead of inverted path IDs, so it must perform several nested structural joins [14] to achieve the answer, and shows poor performance other than the  $Q_3$ , that is the tag-only query.

**Subtree Retrieval.** The start index is the most suitable data structure for subtree retrievals because nodes in a subtree are sequentially ordered. It shows the fastest result (Fig. 5). Nevertheless, both of Xerial and path-start index show almost identical performance to the start index.

**Ancestor Retrieval.** Ancestor query is useful to retrieve parent or ancestor information from some node directly accessed from additional secondary index structures such as the one for traversing IDREF edges, or inverted indexes for text contents. This query needs to find nodes which satisfy  $start < s \wedge e < end$ , where  $(s, e)$  are start and end position of the base node of the query. Fig. 6 shows the performance of the ancestor queries for various positions of base nodes, whose level is 12. The start index processes this query from the root node, and it skips subtrees which are not the ancestor of the base node. The performance of Xerial is stable, because it can eliminate the search space by using a combination of start and end axes. On the other hand, the path-start index breaks the start order down into multiple clusters grouped by path IDs. Consequently,

**Fig. 5.** Suffix-path (left) and subtree (right) query performance**Fig. 6.** Ancestor (left) and sibling (right) query performance

it cannot utilize the tree structure of XML. In addition, it cannot eliminate the search space by using the end values, therefore it is inefficient when the base node of the query has a lot of preceding nodes in the document order. The start index has the same deficit. This result indicates that the ancestor query performance of start and path-start indexes depends on the database size.

**Sibling Retrieval.** Notable usage of sibling node retrievals is to find blank spaces for node insertions, to compute parent-child joins and wild-card(\*) queries. Xerial remarkably outperforms the other indexes (Fig. 6). This is because these indexes except Xerial have difficulty to find nodes in the target level. The start index must repeat searching the tree for a node in the target level with a depth-first traversal, while skipping unrelated descendant nodes occasionally. The path-start index performs this process in every cluster of paths. This descendant skip works well when the target depth of sibling is low; however, as the level becomes deeper, it cannot skip so many descendants and the cost of the B+-tree searches increases. To see this inefficiency, we also provided the result using sequential scan of the entire index, and it shows similar performance to the start index and path-start index for deep levels.

In summary, to efficiently process queries of suffix paths, siblings, subtrees and ancestors, the start-index and the path-start index require additional secondary indexes. For

example, start index should have indexes for level and path, and path-start index needs at least three indexes for end, level, and suffix path. Xerial has the ability to process all of these queries, and the fact it does not use any secondary index is beneficial to the database size and also to the costs of index maintenance due to updates.

## 5 Discussions and Related Works

Although the above experiments show advantages of our methods, we would like to mention some tips that finally lead us to this performance. At first, we used 32-bit integers to represent z-orders, but this implementation performs poorly for every types of queries in the experiments. This is because the 32-bit z-order splits each dimension only to  $2^8$  grids. It is too coarse and results in that too many nodes are assigned the same z-orders; there are many overflowed B+-tree pages and it slows down every search operations. On the other hand, the resolution becomes the finest when every point in the multidimensional space has a unique z-order. However, its bit length might be too long, and such key values will soon fill internal pages of the B+-tree, ending up lowering the B+-tree's branching factors. The optimal resolution is to make *each disk page* have a unique z-order. To achieve this, the UB-tree [17] has to extend the B+-tree implementation.

The use of the UB-tree [17] to index XML documents is proposed in [21]. Its coordinates are combinations of text values, document IDs, and paths and their appearance orders generated from DTDs. Although this method cannot handle suffix path queries etc., the integration of text values is an interesting approach that we do not have mentioned in this paper. Note that, however, if we integrate text values to the index structure, every update to the text values invokes subsequent maintenance of the indexes. Kaushik et. al. proposed efficient algorithms to process queries containing predicates for text values [1]. Their approach assumes text value indexes are maintained separately from structure indexes, so it is more promising in that it can leverage traditional IR technologies to index text contents of XML.

We have not mentioned the updatability of the XML indexes. In fact, integer intervals are weak for updates, since blank space for future node insertions will be exhausted. There are some proposals to make these labels tolerant for node insertions, including ORDPATH [13] etc. As long as we can define the total order on node labels, it is possible to incorporate these labeling strategies to our method.

## 6 Conclusions and Future Work

In this paper, we proposed an efficient method to integrate tree-structure and path-structure indexes for XML. The proposed indexing method provides efficient processing of ancestor, descendant, sibling, suffix path queries etc. In addition, our index structure and multidimensional range query algorithm can be implemented on top of the standard B+-tree. Our experimental results show advantages and disadvantages of query processing due to the indexing methods. Other queries not targeted in this paper are references by using IDREF edges or inverted indexes for the text contents. It is worth investigating to incorporate such additional index structures into Xerial.

## References

1. Kaushik, R., Krishnamurthy, R., Naughton, J.F., Ramakrishnan, R.: On the integration of structure indexes and inverted lists. In: ICDE. (2004) 829
2. Clark, J., DeRose, S.: XML path language (XPath) version 1.0 (1999) available at <http://www.w3.org/TR/xpath>.
3. Li, Q., Moon, B.: Indexing and querying XML data for regular path expressions. In: proc. of VLDB. (2001)
4. Chien, S.Y., Vagena, Z., Zhang, D., Tsotras, V.J., Zaniolo, C.: Efficient structural joins on indexed XML documents. In: proc. of VLDB. (2002)
5. Jiang, H., Wang, W., Lu, H., Yu, J.X.: Holistic twig joins on indexed xml documents. In: proc. of VLDB. (2003)
6. Yi Chen, S.B.D., Zheng, Y.: BLAS: An efficient XPath processing system. In: proc. of SIGMOD. (2004)
7. Jiang, H., Lu, H., Wang, W., Ooi, B.C.: XR-Tree: Indexing xml data for efficient structural joins. In: proc. of ICDE. (2002)
8. Gray, J., Reuter, A.: Transaction Processing - Concepts and Techniques. Morgan Kaufmann (1993)
9. Hellerstein, J.M., Stonebraker, M.: Readings in Database Systems. Forth Edition. MIT Press (2005)
10. Orenstein, J.A., Merrett, T.H.: A class of data structures for associative searching. In: proc. of PODS. (1984)
11. Lawder, J.K., King, P.J.H.: Querying multi-dimensional data indexed using the Hilbert space-filling curve. SIGMOD Record **30**(1) (2001)
12. Sagan, H.: Space-Filling Curves. Springer-Verlag New York, Inc (1994)
13. O'Neil, P., O'Neil, E., pal, S., Cseri, I., Schaller, C.: Orddpaths: Insert-friendly xml node labels. In: proc. of SIGMOD. (2004)
14. Al-Khalifa, S., Jagadish, H.V., Koudas, N., Patel, J.M.: Structural joins: A primitive for efficient XML query pattern matching. In: proc. of ICDE. (2002)
15. Goldman, R., Widom, J.: DataGuides: Enabling query formulation and optimization in semi-structured databases. In: proc. of VLDB. (1997)
16. Milo, T., Suciu, D.: Index structures for path expressions. In: Database Theory - ICDT 99. Volume 1540 of Lecture Notes in Computer Science. (1999)
17. Bayer, R., Markl, V.: The UB-tree: Performance of multidimensional range queries. Technical report (1998)
18. Ramsak, F., Markl, V., Fenk, R., Zirkel, M., Elhardt, K., Bayer, R.: Integrating the UB-tree into a database system kernel. In: proc. of VLDB. (2000)
19. Sleepycat Software: (BerkeleyDB) available at <http://www.sleepycat.com/>.
20. Schmidt, A., Waas, F., Kersten, M., Carey, M.J., manolesch, I., Busse, R.: XMark: A benchmark for XML data management. In: proc. of VLDB. (2002)
21. Bauer, M.G., Ramsak, F., Bayer, R.: Indexing XML as a multidimensional problem. Technical report (2002) TUM-I0203.

# Efficient Support for Ordered XPath Processing in Tree-Unaware Commercial Relational Databases

Boon-Siew Seah<sup>1,2</sup>, Klarinda G. Widjanarko<sup>1,2</sup>, Sourav S. Bhowmick<sup>1,2</sup>,  
Byron Choi<sup>1</sup>, and Erwin Leonardi<sup>1,2</sup>

<sup>1</sup> School of Computer Engineering, Nanyang Technological University, Singapore

<sup>2</sup> Singapore-MIT Alliance, Nanyang Technological University, Singapore

{8212123145823, klarinda, assourav, kkchoi, lerwin}@ntu.edu.sg

**Abstract.** In this paper, we present a novel ordered XPath evaluation in *tree-unaware* RDBMS. The novelties of our approach lies in the followings. (a) We propose a novel XML storage scheme which comprises *only* leaf nodes, their corresponding data values, order encodings and their root-to-leaf paths. (b) We propose an algorithm for mapping ordered XPath queries into SQL queries over the storage scheme. (c) We propose an optimization technique that enforces all mapped SQL queries to be evaluated in a “left-to-right” join order. By employing these techniques, we show, through a comprehensive experiment, that our approach not only scales well but also performs better than some representative tree-unaware approaches on more than 65% of our benchmark queries with the highest observed gain factor being 1939. In addition, our approach reduces significantly the performance gap between tree-aware and tree-unaware approaches and even outperforms a state-of-the-art tree-aware approach for certain benchmark queries.

## 1 Introduction

Current approaches for evaluating XPath expressions in relational databases can be arguably categorized into two representative types. They either resort to encoding XML data as tables and translating XML queries into relational queries [12346811] or store XML data as a rich data type and process XML queries by enhancing the relational infrastructure [5]. The former approach can further be classified into two representative types. Firstly, a host of work on processing XPath queries on *tree-unaware* relational databases has been reported [368] – these approaches do not modify the database kernels. Secondly, there have been several efforts on enabling relational databases to be *tree-aware* by invading the database kernel to implement XML support [12411]. It has been shown that the latter approaches appear scalable and, in particular, perform orders of magnitude faster than some tree-unaware approaches [14].

In this paper, we focus on supporting *ordered* XPath evaluation in a *tree-unaware* relational environment. There is a considerable benefit in such an approach with respect to portability and ease of implementation on top of an off-the-shelf RDBMS. Although a diverse set of strategies for evaluating XML queries in tree-unaware relational environment have been recently proposed, few have undertaken a comprehensive study on evaluating ordered XPath queries. Tatarinov *et al.* [9] is the first to show that it is indeed possible to support ordered XPath queries in relational databases. However, this

approach does not scale well with large XML documents. In fact, as we shall show in Section 7, the GLOBAL-ORDER approach in [9] failed to return results for 20% of our benchmark queries on 1GB dataset in 60 minutes. Furthermore, this approach resorts to manual tuning of the relational optimizer when it failed to produce good query plans. Although such a manual tuning approach works, it is a cumbersome solution.

In this paper, we address the above limitations by proposing a novel scheme for ordered XPATH query processing. Our storage strategy is built on top of SUCXENT++ [6], by extending it to support efficient processing of ordered axes and predicates. SUCXENT++ is designed primarily for query-mostly workloads. We exploit SUCXENT++’s strategy to store leaf nodes, their corresponding data values, auxiliary encodings and root-to-leaf paths. In contrast, some approaches, *e.g.*, [4][11], explicitly store information for all nodes of an XML document. Specifically, the followings remark the novelties of our storage scheme. (1) For each level of an XML document, we store an attribute called RValue which is an enhancement of the original RValue, proposed in [6], for processing recursive XPATH queries. (2) For each leaf node we store three additional attributes namely BranchOrder, DeweyOrderSum and SiblingSum. These attributes are the foundation for our ordered XPATH processing. The key features of these attributes are that they enable us (a) to compare the order between non-leaf nodes by comparing the order between their *first descendant leaf* nodes only; and (b) to determine the nearest common ancestor of two leaf nodes efficiently. As a result, it is not necessary to store the order information of non-leaf nodes. Furthermore, given any pair of nodes, these attributes enable us to evaluate position-based predicates efficiently.

As highlighted in [9], relational optimizers may sometimes produce poor query plans for processing XPATH queries. In this paper, we undertake a novel strategy to address this issue. As opposed to manual tuning efforts, we propose an *automatic* approach to enforce the optimizer to replace previously generated poor plans with probably better query plans, as verified by our experiments. Unlike tree-aware schemes, our technique is *non-invasive* in nature. That is, it can easily be incorporated *without modifying the internals* of relational optimizers. Specifically, we enforce a relational optimizer to follow a “*left-to-right*” join order and enforce the relational engine to evaluate the mapped SQL queries according to the XPATH steps specified in the query. The good news is that this technique can select better plans for the majority of our benchmark queries across all benchmark datasets. As we shall see in Section 7, the performance of previously-inefficient queries in SUCXENT++ is significantly improved. The highest observed gain factor is 59. Furthermore, queries that failed to finish in 60 minutes were able to do so now, in the presence of such a join-order enforcement. This is indeed stimulating as it shows that some sophisticated internals of relational optimizers not only are irrelevant to XPATH processing but also often confuse XPATH query optimization in relational databases. Overall a “join-order conscious” SUCXENT++ significantly outperforms both GLOBAL-ORDER and SHARED-INLINING[8] in at least 65% of the benchmark queries with the highest observed gain factors being 1939 and 880, respectively. To the best of our knowledge, this is the first effort on exploiting a non-invasive automatic technique to improve query performance *in the context of XPATH evaluation in relational environment*.

Recently, [1] showed that MONETDB is among the most efficient and scalable tree-aware relational-based XQuery processor and outperforms the current generation of XQuery systems significantly. Consequently, we investigated how our proposed technique compared to MONETDB. Our study revealed some interesting results. First, although MONETDB is 11-164 and 3-74 times faster than GLOBAL-ORDER and SHARED-INLINING, respectively, for the majority of the benchmark queries, this performance gap is significantly reduced when MONETDB is compared to SUCXENT++. Our results show that not only MONETDB is now 1.3-16 times faster than SUCXENT++ with join-order enforcement but surprisingly our approach is faster than MONETDB for 33% of benchmark queries! Additionally, MONETDB (Win32 builds) failed to shred 1GB dataset as it is vulnerable to the virtual memory fragmentation in Windows environment. This is in contrary to the results in [1] where MONETDB was built on top of Linux 2.6.11 operating system (8GB RAM), using a 64-bit address space, and was able to efficiently shred 11GB dataset.

## 2 Related Work

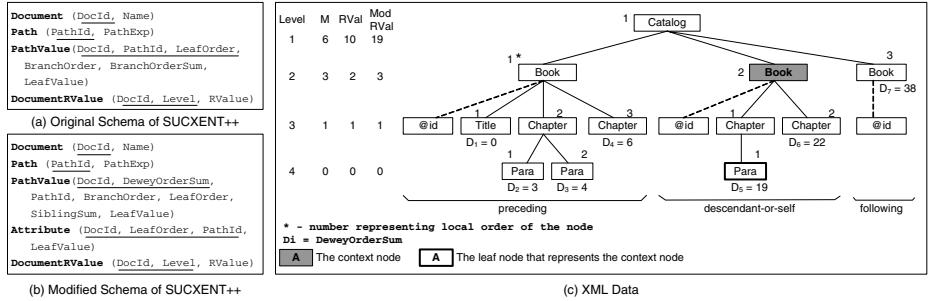
Most of the previous tree-unaware approaches, except [9], focused on proposing efficient evaluation for children and descendant-or-self axes and positional predicates in XPATH queries. In this paper, the main focus is on the evaluation for following, preceding, following-sibling, and preceding-sibling axes as well as *position-based* and *range* predicates. All previous approaches, reported query performance on small/medium XML documents – smaller than 500 MB. We investigate query performance on large synthetic and real datasets. This gives insights on the scalability of the state-of-the-art tree-unaware approaches for ordered XML processing.

Compared to the tree-aware schemes [12411], our technique is *tree-unaware* in the sense that it can be built on top of any commercial RDBMS without modifying the database kernel. The approaches in [211] do not provide a systematic and comprehensive effort for processing ordered XPATH queries. Although the scheme presented in [124] can support ordered axes, no comprehensive performance study has demonstrated with a variety of ordered XPATH queries. Furthermore, these approaches did not exploit the “left-to-right” join order technique to improve query plan selection.

In [9], Tatarinov *et al.* proposed the first solution for supporting ordered XML query processing in a relational database. A modified EDGE table [3] was the underlying storage scheme. They described three order encoding methods: *global*, *local*, and *dewey* encodings. The best query performance was achieved with the *global* encoding for query-mostly workloads and with *dewey* encoding for a mix of queries and updates. Our focus differs from the above approach in the following ways. First, we focus on query-mostly workloads. Second, we consider a novel order-conscious storage scheme that is more space- and query-efficient and scalable when compared to the *global* encoding.

## 3 Background on SUCXENT++

Our approach for ordered XPATH processing relies on the SUCXENT++ approach [6]. We begin our discussion by briefly reviewing the storage scheme of SUCXENT++.



**Fig. 1.** Example of XML data and SUCXENT++ schema

Foremost, in the rest of the paper, we always assume *document order* in our discussions. The SUCXENT++ schema is shown in Figure II(a). Document stores the document identifier DocId and the name Name of a given input XML document  $T$ . We associate each distinct (root-to-leaf) path appearing in  $T$ , namely PathExp, with an identifier PathId and store this information in Path table. For each leaf node  $n$  in  $T$ , we shall create a tuple in the PathValue table. We now elaborate the meaning of the attributes of this relation.

Given two leaf nodes  $n_1$  and  $n_2$ ,  $n_1.\text{LeafOrder} < n_2.\text{LeafOrder}$  iff  $n_1$  precedes  $n_2$ .  $\text{LeafOrder}$  of the first leaf node in  $T$  is 1 and  $n_2.\text{LeafOrder} = n_1.\text{LeafOrder} + 1$  iff  $n_1$  is a leaf node immediately preceding  $n_2$ . Given two leaf nodes  $n_1$  and  $n_2$  where  $n_1.\text{LeafOrder} + 1 = n_2.\text{LeafOrder}$ ,  $n_2.\text{BranchOrder}$  is the level of the nearest common ancestor of  $n_1$  and  $n_2$ . That is,  $n_1$  and  $n_2$  intersect at the  $\text{BranchOrder}$  level. The data value of  $n$  is stored in  $n.\text{LeafValue}$ .

To discuss BranchOrderSum and RValue, we introduce some auxiliary definitions. Consider a sequence of leaf nodes  $C: \langle n_1, n_2, n_3, \dots, n_r \rangle$  in  $T$ . Then,  $C$  is a *k-consecutive leaf nodes* of  $T$  iff (a)  $n_i.\text{BranchOrder} \geq k$  for all  $i \in [1, r]$ ; (b) If  $n_1.\text{LeafOrder} > 1$ , then  $n_0.\text{BranchOrder} < k$  where  $n_0.\text{LeafOrder} + 1 = n_1.\text{LeafOrder}$ ; and (c) If  $n_r$  is not the last leaf node in  $T$ , then  $n_{r+1}.\text{BranchOrder} < k$  where  $n_r.\text{LeafOrder} + 1 = n_{r+1}.\text{LeafOrder}$ . A sequence  $C$  is called a *maximal k-consecutive leaf nodes* of  $T$ , denoted as  $M_k$ , if there does not exist a  $k$ -consecutive leaf nodes  $C'$  and  $|C| < |C'|$ .

Let  $L_{max}$  be the largest level of  $T$ . Then, RValue of level  $\ell$ , denoted as  $R_\ell$ , is 1 if  $\ell = L_{max}$ . Otherwise,  $R_\ell = R_{\ell+1} \times |M_{\ell+1}| + 1$ . Now we are ready to define the BranchOrderSum attribute. Let  $N$  to be the set of leaf nodes preceding a leaf node  $n$ .  $n.BranchOrderSum$  is 0 if  $n.LeafOrder = 1$  and  $\sum_{m \in N} R_{m.BranchOrder}$  otherwise.

Based on the definitions above, Prakash *et al.* [6] defined Property II (below) which is essential to determine ancestor-descendant relationships efficiently.

*Property 1.* Given two leaf nodes  $n_1$  and  $n_2$ ,  $|n_1.\text{BranchOrderSum} - n_2.\text{BranchOrderSum}| < R_\ell$  implies the nearest common ancestor of  $n_1$  and  $n_2$  is at a level greater than  $\ell$ .  $\square$

## 4 Extensions of SUCXENT++

To support ordered XML queries, the order information of nodes must be captured in the XML storage scheme. Unfortunately the LeafOrder and BranchOrderSum attributes only

encode the global order of all leaf nodes. Since (order) information of non-leaf nodes is not explicitly stored, it must be derived from the attributes of leaf nodes. We now present how the original SUCXENT++ schema is extended to process ordered XPath queries efficiently. The modified schema is shown in Figure 11(b).

#### 4.1 Attribute Table

The PathValue table originally stored information related to both element and attribute nodes. However, to avoid mixing the order of element and attribute nodes, we separate the attribute nodes into Attribute table. The Attribute table consists of the following columns: DocId, LeafOrder, PathId, LeafValue. As we shall see later, a non-leaf node can be represented by the first descendant leaf nodes. Therefore, an attribute node is identified by DocId and LeafOrder of its parent node and its PathId.

#### 4.2 Modified RValue Attribute

Conceptually, RValue is used to encode the level of the nearest common ancestor of any pairs of leaf nodes. To ensure a property like Property 1 holds after modifications, intuitively, we “magnify” the gap between RValues, as shown in Definition 1. Relative order information is then captured in these gaps.

**Definition 1 [ModifiedRValue].** Let  $L_{max}$  be the largest level of an XML tree  $T$ . **ModifiedRValue** of level  $\ell$ , denoted as  $R'_\ell$ , is defined as follows: (i) If  $\ell = L_{max} - 1$  then  $R'_\ell = 1$  and  $|M_\ell| = 1$ ; (ii) If  $0 < \ell < L_{max} - 1$  then  $R'_\ell = 2R'_{\ell+1} \times |M_{\ell+1}| + 1$ .  $\square$

To ensure the evaluation of queries other than ordered XPATH queries is not affected by the above modifications, the RValue attribute in DocumentRValue stores  $\frac{R'_\ell - 1}{2} + 1$  instead of  $R'_\ell$ .

#### 4.3 DeweyOrderSum and SiblingSum Attributes

Next, we define the first attribute related to ordered XPATH processing. Consider the path query `/catalog/book[1]/chapter[1]` and Figure 11(c). Since only leaf nodes are stored in the PathValue table, the new attribute DeweyOrderSum of leaf nodes captures order information of the non-leaf nodes. At first glance, a simple representation of the order information could be a Dewey path. For instance, the Dewey path of the first chapter node of the first book node is “1.1.2”. However, using such Dewey paths has two major drawbacks. Firstly, string matching of Dewey paths can be computationally expensive. Secondly, simple lexicographical comparisons of two Dewey paths may not always be accurate [9]. Hence, we define DeweyOrderSum for this purpose:

**Definition 2 [DeweyOrderSum].** Consider an XML document  $T$  and a leaf node  $n$  at level  $\ell$  in  $T$ .  $\text{Ord}(n, k) = i$  iff  $a$  is either an ancestor of  $n$  or  $n$  itself;  $k$  is the level of  $a$ ; and  $a$  is the  $i$ -th child of its parent. DeweyOrderSum of  $n$ ,  $n$ .DeweyOrderSum, is defined as  $\sum_{j=2}^{\ell} \Phi(j)$  where  $\Phi(j) = [\text{Ord}(n, j) - 1] \times R'_{j-1}$ .  $\square$

For example, consider the rightmost `chapter` node in Figure 1(c) which has a Dewey path “1.2.2”. `DeweyOrderSum` of this node is:  $n.\text{DeweyOrderSum} = (\text{Ord}(n, 2) - 1) \times R'_1 + (\text{Ord}(n, 3) - 1) \times R'_2 = 1 \times 19 + 1 \times 3 = 22$ . Note that `DeweyOrderSum` is not sufficient to compute position-based predicates with `QName` name tests, e.g., `chapter[2]`. Hence, the `SiblingSum` attribute is introduced to the `PathValue` table.

**Definition 3 [SiblingSum].** Consider an XML document  $T$  and a leaf node  $n$  at level  $\ell$  in  $T$ .  $\text{Sibling}(n, k) = i$  iff  $a$  is either an ancestor of  $n$  or  $n$  itself;  $k$  is the level of  $a$ ; and the  $i$ -th  $\tau$ -child of its parent ( $\tau$  is the tag name of  $a$ ).  $\text{SiblingSum}$  of  $n$ ,  $n.\text{SiblingSum}$ , is  $\sum_{j=2}^{\ell} \Psi(j)$  where  $\Psi(j) = [\text{Sibling}(n, j) - 1] \times R_{j-1}$ .  $\square$

`SiblingSum` encodes the local order of nodes which are with the same tag name of  $n$ , namely same-tag-sibling order. For example, consider the children of the first `book` element in Figure 1(c). The local orders of `title` and the first and second `chapter` nodes are 1, 2 and 3, respectively. On the other hand, the same-tag-sibling order of these nodes are 1, 1 and 2, respectively.

#### 4.4 Preservation of SUCXENT++’s Features

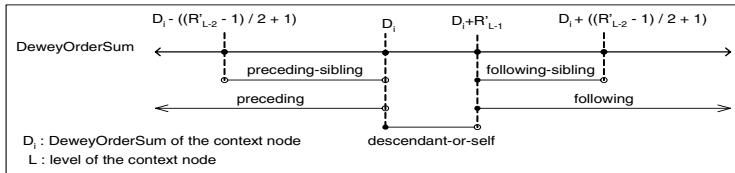
The above modifications do not adversely affect the document reconstruction process and efficient evaluation of non-ordered XPATH queries, as discussed in [6]. Recall that given a pair of leaf nodes, Property 1 was used in [6] to efficiently determine the nearest common ancestor of the nodes. Since we have modified the definition of `RValue` and replaced the `BranchOrderSum` attribute with the `DeweyOrderSum` attribute, this property is not applicable to the extended SUCXENT++ scheme. It is necessary to ensure that a corresponding property holds in the extended system.

**Theorem 1.** Let  $n_1$  and  $n_2$  be two leaf nodes in an XML document. If  $\frac{R'_{\ell+1}-1}{2} + 1 < |n_1.\text{DeweyOrderSum} - n_2.\text{DeweyOrderSum}| < \frac{R'_\ell-1}{2} + 1$  then the level of the nearest common ancestor of  $n_1$  and  $n_2$  is  $\ell + 1$ .  $\square$

Due to space constraints, the proofs and examples of the theorems and propositions discussed in this paper are given in [7].

## 5 Ordered XPath Processing

Our strategy for comparing the order of non-leaf nodes is based on the following observation. If node  $n_0$  precedes (resp. follows) another node  $n_1$ , then descendants of  $n_0$  must also precede (resp. follow) the descendants of  $n_1$ . Therefore, instead of comparing the order between non-leaf nodes, we compare the order between *their descendant leaf nodes*. For this reason, we define the *representative leaf node* of a non-leaf node  $n$  to be its first descendant leaf node. Note that the `BranchOrder` attribute records the level of the nearest common ancestor of two consecutive leaf nodes. Let  $C$  be the sequence of descendant leaf nodes of  $n$  and  $n_1$  be the first node in  $C$ . We know that the nearest common ancestor of any two consecutive nodes in  $C$  is also a descendant of node  $n$ . This implies (1) except  $n_1$ , `BranchOrder` of a node in  $C$  is at least the level of node  $n$ .



**Fig. 2.** Relationship between DeweyOrderSum and RValue

and (2) the nearest common ancestor of  $n_1$  and its immediately preceding leaf node is not a descendant of node  $n$ . Therefore, BranchOrder of  $n_1$  is always smaller than the level of  $n$ . We summarize this property in Property 2.

*Property 2.* Let  $n$  be a non-leaf node at level  $\ell$  and  $C = \langle n_1, n_2, n_3, \dots, n_r \rangle$  be the sequence of descendant leaf nodes of  $n$  in document order. Then,  $n_1.\text{BranchOrder} < \ell$  and  $n_i.\text{BranchOrder} \geq \ell$ , where  $i \in (1, r]$ .  $\square$

**Definition 4 [DeweyOrderSum of non-leaf nodes].** Let  $S = \langle i_1, i_2, i_3, \dots, i_{r_1} \rangle$  be a sequence of non-leaf sibling nodes of a non-leaf node  $i_0$  in document order. Let  $C = \langle n_1, n_2, \dots, n_{r_2} \rangle$  be the sequence of leaf nodes of  $S$  and  $n_{j_2}$  is denoted as the first descendant leaf node of  $i_{j_1}$ . Then,  $i_{j_1}.\text{DeweyOrderSum} = n_{j_2}.\text{DeweyOrderSum}$ .  $\square$

In the above definition, DeweyOrderSum of a leaf node is *conceptually* propagated to its ancestor nodes. Consequently, the following proposition holds.

**Proposition 1.** Let  $C = \langle n_1, n_2, n_3, \dots, n_r \rangle$  be a sequence of sibling nodes. Consider  $n_i$  where  $1 < i \leq r$  and the level of  $n_i$  is  $\ell$ , where  $\ell > 1$ . Let  $m$  be  $n_i$  or a descendant of  $n_i$ . Then,  $n_1.\text{DeweyOrderSum} + [\text{Ord}(n_i) - \text{Ord}(n_1)] \times R'_{\ell-1} \leq m.\text{DeweyOrderSum} < n_1.\text{DeweyOrderSum} + [(\text{Ord}(n_i) - \text{Ord}(n_1)) + 1] \times R'_{\ell-1}$  where  $\text{Ord}(n_i)$  and  $\text{Ord}(n_1)$  are the local order of  $n_i$  and  $n_1$ , respectively.  $\square$

By using the above proposition, we can compare the order of two non-leaf nodes without evaluating every sibling nodes in the sequence. Similar propositions for SiblingSum can be established in a straightforward manner.

## 5.1 Support for Ordered XPath Queries

We now present how various types of ordered XPATH queries are supported by the modified SUCXENT++. Due to space constraints, we only focus on how DeweyOrderSum and ModifiedRValue are used for query processing. Similar technique can be applied to evaluations with SiblingSum.

**Position predicates.** Position-based predicates, *i.e.*, predicates of the form  $\text{position}()=i$ , select the node at the  $i$ -th position of the sequence of *inner focus context nodes*. We propose to compute the  $i$ -th node without evaluating every node in the sequence by applying Proposition 1. For example, suppose  $n_1$  be the first book node of the sequence of book nodes (the context nodes) in Figure 1c). Observe that  $n_1.\text{DeweyOrderSum} = 0$  as its representative leaf node is the first leaf node of the XML tree. We now employ the inequality in Proposition 1 to select a sibling node, *e.g.*, the second book

node  $n_2$ . Here,  $\text{Ord}(n_2) = 2$ ,  $\ell = 2$ ,  $R'_1 = 19$ , and  $n_1.\text{DeweyOrderSum} = 0$ . Then,  $0 + 1 \times 19 \leq n_2.\text{DeweyOrderSum} < 0 + 2 \times 19 \Rightarrow 19 \leq n_i.\text{DeweyOrderSum} < 38$ . The nodes in this range are the descendant leaf nodes of  $n_2$ . Such simple arithmetic calculations can be efficiently implemented in a relational database.

The range operator, e.g., `[position()=2 TO 10]`, can be easily handled in a similar fashion.  $\text{fn : last()}$  can be computed by first determining all sibling nodes that satisfy the specific path and then finding the node with the largest  $\text{DeweyOrderSum}$ .

**Following and preceding axes.** `following` axis selects all nodes which follow the context node excluding the descendants of the context node. `preceding` axis, on the other hand, selects all nodes which precede the context node excluding the ancestors of the context node. Similar to position predicates, we summarize a property of  $\text{DeweyOrderSum}$  to facilitate efficient processing of these axes. Proofs and additional examples are given in [7].

**Proposition 2.** *Let  $n_a$  and  $n_b$  be two nodes in the XML tree  $T$  and  $n_b$  is a context node at level  $\ell_b$  where  $\ell_b > 1$ . Then, the following statements hold:*

1.  $n_a.\text{DeweyOrderSum} \geq n_b.\text{DeweyOrderSum} + R'_{\ell_b-1}$  if and only if  $n_a$  follows  $n_b$  and is not a descendant of  $n_b$ ;
2. Similarly,  $n_a.\text{DeweyOrderSum} < n_b.\text{DeweyOrderSum}$  if and only if  $n_a$  precedes  $n_b$  and  $n_a$  is neither a descendant nor an ancestor of  $n_b$ .

□

**Following-sibling and preceding-sibling axes.** `following-sibling` axis selects the children of the context node's parent that occur after the context node in document order whereas `preceding-sibling` axis selects the children of the context node's parent that occur before the context node in document order. Support for `following-sibling` (resp. `preceding-sibling`) axis can be achieved with an additional constraint on the `following` (resp. `preceding`) axis – the selected nodes must be siblings of the context node.

**Proposition 3.** *Let  $n_a$  and  $n_b$  be two nodes in the XML tree  $T$  and  $n_b$  is the context node at level  $\ell_b$  where  $\ell_b > 2$ . Then, the following statements hold:*

1.  $n_b.\text{DeweyOrderSum} + R'_{\ell_b-1} \leq n_a.\text{DeweyOrderSum} < n_b.\text{DeweyOrderSum} + (R'_{\ell_b-2} - 1)/2 + 1$  and if and only if  $n_a$  is a sibling of  $n_b$  and  $n_a$  follows  $n_b$ .
2.  $n_b.\text{DeweyOrderSum} - (R'_{\ell_b-2} - 1)/2 - 1 < n_a.\text{DeweyOrderSum} < n_b.\text{DeweyOrderSum}$  if and only if  $n_a$  is a sibling of  $n_b$  and  $n_a$  precedes  $n_b$ .

□

The above proposition can be illustrated with the following example. Suppose we evaluate the `following-sibling` axis on the first `title` node  $n_t$  in Figure II(c). Here  $n_t.\text{DeweyOrderSum} = 0$ ,  $\ell = 3$ ,  $R'_1 = 19$ , and  $R'_2 = 3$ . Denote  $N$  to be the nodes reachable via the `following-sibling` axis from  $n_t$ . Using Proposition 3,  $0 + 3 \leq n_k.\text{DeweyOrderSum} < 0 + (19 - 1)/2 + 1$  where  $n_k \in N$ . That is,  $3 \leq n_k.\text{DeweyOrderSum} < 10$ . Hence, the second ( $\text{DeweyOrderSum} = 3$ ) and the third ( $\text{DeweyOrderSum} = 6$ ) chapters are in this range.

We illustrate Proposition 2 and Proposition 3 with Figure 2. An example can be found in Figure II(c).

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre><b>processPathExpr (XPath)</b> 01 for every step in the XPath { 02   if (step.getAxis() == CHILD and 03       step.hasPredicate() == FALSE) 04     currentPath.add(nametest, step.getAxis()) 05   else 06     from_sql.add("PathValue as V") 07     if(currentPath.level() &gt; 1) { 08       where_sql.add("V.pathid in currentPath.getPathId()") 09       where_sql.add("V.branchOrder &lt; currentPath.level()") 10     } 11     processAxis(step, currentPath) 12     processPredicate(step, currentPath) 13   } 14   if (step.isLast() and currentPath.needUpdate()) { 15     from_sql.add("PathValue as V") 16     where_sql.add("V.pathid in currentPath.getPathId()") 17   } 18   select_sql.add("V.leafValue, V.leafOrder, ... ") 19 return select_sql + from_sql + where_sql +   where_sql.unionWithAttribute()</pre> | <pre><b>processAxis (step, currentPath)</b> 01 switch (step.getAxis())( 02   child: 03     where_sql.add("V.DeweyOrderSum BETWEEN   V_{-1}.DeweyOrderSum + RValue(currentPath.level() - 1) + 1 AND   V_{-1}.DeweyOrderSum + RValue(currentPath.level() - 1) - 1 ") 04   following: 05     where_sql.add("V_i.DeweyOrderSum &gt;=   V_{-1}.DeweyOrderSum + 2 * RValue(currentPath.level() - 1) ") 06   preceding: 07     where_sql.add("V_i.DeweyOrderSum &lt; V_{-1}.DeweyOrderSum ") 08   following-sibling: 09     where_sql.add("V_i.DeweyOrderSum BETWEEN   V_{-1}.DeweyOrderSum + 2 * RValue(currentPath.level() - 1) + 1 AND   V_{-1}.DeweyOrderSum + RValue(currentPath.level() - 1) - 1 ") 10   preceding-sibling: 11     where_sql.add("V_i.DeweyOrderSum BETWEEN   V_{-1}.DeweyOrderSum - RValue(currentPath.level() - 1) + 1 AND   V_{-1}.DeweyOrderSum - 1 ") 12 ) 13 currentPath.add(nametest, step.getAxis())</pre> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

(a)The processPathExpr Algorithm

(b)The processAxis Algorithm

**Fig. 3.** Procedure processPathExpr and Procedure processAxis

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre><b>processPredicate (step, currentPath)</b> 01 switch (step.getAxis()) { 02   CHILD: 03     n_from = step.getPredicateFrom() - 1 04     n_to = step.getPredicateTo() 05   FOLLOWING: 06     n_from = step.getPredicateFrom() 07     n_to = step.getPredicateTo() + 1 08   PRECEDING: 09     n_from = - step.getPredicateFrom() 10     n_to = - step.getPredicateTo() + 1 11   } 12 switch (step.getAxis().getAxisType()) 13   position based predicate without name test: 14     where_sql.add("V.DeweyOrderSum BETWEEN   V_{-1}.DeweyOrderSum + n_from *   (2 * RValue(currentPath.level()) - 1) AND   V_{-1}.DeweyOrderSum + n_to *   (2 * RValue(currentPath.level()) - 1) - 1 ") 15   position based predicate with name test: 16     where_sql.add("V_i.SiblingSum BETWEEN   V_{-1}.RValue(currentPath.level()) - 1 AND   V_{-1}.RValue(currentPath.level()) - 1 - 1 ") 17 }</pre> | <pre>01 <b>WITH</b> V (leafValue, pathID, branchOrder, DeweyOrderSum,   DocId, LeafOrder AS 02 <b>SELECT</b> DISTINCT leafValue, V2.pathID, V2.branchOrder,   FROM PathValue V1, PathValue V2 03 <b>WHERE</b> V1.docId = V2.docId AND V1.level in (5,4,3,2) 04 <b>AND</b> V1.SiblingSum <b>BETWEEN</b> 05   0 + 1 * (2 * 10 - 1) <b>AND</b> 06   0 + 1 * (2 * 10 - 1) - 1 07 <b>AND</b> V1.branchOrder = V2.branchOrder 08 <b>AND</b> V2.docId = V1.docId 09 <b>AND</b> V2.pathid in (5,4,3,2) 10 <b>AND</b> V2.DeweyOrderSum &gt;= V1.DeweyOrderSum + 2 * 10 - 1 11 <b>AND</b> V2.DeweyOrderSum <b>BETWEEN</b> 12   V1.DeweyOrderSum + 1 * (2 * 10 - 1) <b>AND</b> 13   V1.DeweyOrderSum + 2 * (2 * 10 - 1) - 1 14 <b>)</b> 15 <b>SELECT</b> V.* , 1 AS Attr 16 <b>FROM</b> V 17 <b>UNION</b> ALL 18 <b>SELECT</b> A.leafValue, A.pathID, V.branchOrder, V.DeweyOrderSum,   A.DocId, A.LeafOrder, 0 AS Attr 19 <b>FROM</b> Attribute A 20 <b>WHERE</b> A.DocId = V.DocId <b>AND</b> A.LeafOrder = V.LeafOrder 21 <b>AND</b> A.Pathid in (1) 22 <b>ORDER</b> BY DocId, DeweyOrderSum, Attr</pre> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

(a)The processPredicate Algorithm

(b) SQL Example

**Fig. 4.** Procedure processPredicate and SQL example

## 5.2 Ordered XPath Query Translation Algorithm

Based on the properties defined in the previous subsection, we present an algorithm, shown in Figures 3 and 4, for generating SQL from ordered XPATH queries. Our algorithm assumes an XPATH expression is represented as a sequence of steps where a step may be associated with predicates. A SQL statement consists of three clauses: *select\_sql*, *from\_sql* and *where\_sql*. We assume that a clause has an *add()* method which encapsulates some simple string manipulations and simple SUCXENT++ joins for constructing valid SQL statements. In addition to preprocessing PathId as mentioned in [6], for a single XML document, we also preprocess RValue to reduce the number of joins. The translation consists of three main procedures.

**processPathExpr (Figure 3(a)):** It analyzes the steps of an input XPATH expression (Line 01) and outputs a SQL statement. If the step consists of a child axis only (Lines 02-03), then we simply maintain a global variable *currentPath* which records the simple downward path from the root to the context nodes.<sup>1</sup> Otherwise, when the step involves ordered predicates/other axes, we add predicates which select a superset of the next context nodes (Lines 05-09) and then call *processAxis* and *processPredicate* (Lines 10-11) with *currentPath* to obtain the next context

<sup>1</sup> The details for maintaining *currentPath* is simple but lengthy. For simplicity, we omitted such discussions.

nodes. We add predicates in Lines 08 to determine the representative nodes of the context nodes. Finally, we collect the final results (Line 19).

**processAxis (Figure 3(b)):** This procedure translates a step, together with *currentPath*, based on the step type (Line 01). Lines 02-03, 04-07 and 08-11 encode Theorem 1, Proposition 2 and Proposition 3 respectively.

**processPredicate (Figure 4(a)):** This procedure mainly translates position predicates. Lines 01-11 determine the range of position specified by the predicate. Given these, Lines 12-17 implement Proposition 1.

We now illustrate the details of the translation algorithms with an example related to the translation of position-based predicates. Please refer to [7] for more examples. Consider the path expression `/catalog/book[2]/following-sibling::*` [1]. The translated SQL is shown in Figure 4(b). `/catalog/book[2]` is translated into Lines 05-07. `/following-sibling::*` is translated into Lines 08-10, and `* [1]` is translated into Line 11. Lines 13-14 and 16-19 are used to retrieve the resulting element nodes and their attribute nodes, respectively. The last line is to sort the result nodes in document order.

## 6 Join Order Enforcement

Due to the tree-unaware nature of the underlying relational storage scheme as well as the lack of appropriate XML statistics, relational optimizers may generate inefficient query plans. In order to address this problem, some approaches have resorted to manual tuning of query plans [9] while others invade the database kernel to make it tree-aware [12]. The former approach has not been scalable as it requires significant human intervention whereas the later approach may require non-trivial modifications of the internals of a RDBMS. In this section, we propose a simple yet effective technique to generate better query plans automatically *without invading the database kernel*.

As discussed in Section 5.2 in order to evaluate an (ordered) XPATH query in SUCXENT++, each XPATH axis is translated into a join between the PathValue table and intermediate results (*i.e.*, the context nodes). For example, in Figure 4(b), PathValue V1 returns the representative nodes of the context nodes to calculate PathValue V2. Due to the lack of tree awareness, the relational optimizer is not capable of transforming the order of joins intelligently. Consequently, it may generate poor join order that typically requires caching large intermediate results in the database bufferpool. This is particularly important to NL joins, where large and deep loops are prohibitive. For example, the first few joins of a “right-to-left” join order may easily yield a large number of context nodes. To respond to this, we propose to enforce a “left-to-right” join order on the translated SQL query. Also, this evaluation order “naturally corresponds” to the order of XPATH steps specified in the XPATH expression. By employing this technique, the relational optimizer does not explore the large number of permutations of join order. We apply join order if the translated SQL query involves more than one PathValue relation. In addition, if the PathValue table appears in the SQL query only once, we let the relational optimizer to decide the plan for the join between the PathValue table and the Attribute table.

The above enforcement can easily be implemented by *query hints* in commercial databases. Regarding our implementation, we use `OPTION(FORCE ORDER)` to

| ID     | Total Number |           |            | Size (MB) | Max Depth | ID | Query                                      | Res. Card. |
|--------|--------------|-----------|------------|-----------|-----------|----|--------------------------------------------|------------|
|        | Node         | Attribute | Total      |           |           |    |                                            |            |
| DC10   | 225,234      | 15,000    | 240,234    | 10.3      | 8         | D1 | /dblp/*[100000]/author                     | 2          |
| DC100  | 2,242,200    | 150,000   | 2,392,200  | 103.3     | 8         | D2 | /dblp/article/author[2]                    | 190,838    |
| DC1000 | 22,442,612   | 1,500,000 | 23,942,612 | 1033.3    | 8         | D3 | /dblp/*[600000]/pages/preceding-sibling::* | 6          |
| DBLP   | 8,222,945    | 1,665,930 | 9,888,875  | 335       | 6         | D4 | /dblp/*[600000]/pages/following-sibling::* | 5          |

(a) Features of Dataset

| ID | Query                                                        | Res. Card. (10MB) | Res. Card. (100MB) | Res. Card. (1000MB) | ID | Query                                              | Res. Card. (10MB) | Res. Card. (100MB) | Res. Card. (1000MB) |
|----|--------------------------------------------------------------|-------------------|--------------------|---------------------|----|----------------------------------------------------|-------------------|--------------------|---------------------|
| Q1 | /catalog/item[1000]                                          | 66                | 119                | 74                  | Q5 | /catalog/*[1500]/publisher/following-sibling::*    | 30                | 34                 | 34                  |
| Q2 | /catalog/*[1000]                                             | 66                | 119                | 74                  | Q6 | /catalog/*[1500]/publisher/following-sibling::*[5] | 7                 | 7                  | 7                   |
| Q3 | /catalog/item[position()=1000 to 10000]/*[position()=2 to 7] | 104,272           | 626,812            | 627,200             | Q7 | /catalog/*[1500]/publisher/preceding-sibling::*    | 21                | 37                 | 54                  |
| Q4 | /catalog/item[*[position()=1000 to 10000]/authors/author     | 65,161            | 392,930            | 393,350             | Q8 | /catalog/*[1500]/publisher/preceding-sibling::*[2] | 19                | 35                 | 52                  |

(b) Benchmark queries for DC10, DC100, and DC1000

| ID | Query                           | Res. Card. (10MB) | Res. Card. (100MB) | Res. Card. (1000MB) |
|----|---------------------------------|-------------------|--------------------|---------------------|
| Q9 | /catalog/*[X]//following::title | 250               | 2,500              | 25,000              |

(c) Benchmark queries for DBLP

**Fig. 5.** Dataset and Benchmark Queries

implement the above technique in SUCXENT++. The strength of this approach lies in its simplicity in implementing on any commercial RDBMS that supports query hints.

## 7 Performance Study

In this section, we present the results of our performance evaluation on our proposed approach, a tree-unaware schema-oblivious approach (GLOBAL-ORDER [9]), a tree-unaware schema-conscious approach (SHARED-INLINING [8]), and a tree-aware approach (MonetDB [11]). Prototypes for modified SUCXENT++ (denoted as SX), SUCXENT++ with join order enforcement (denoted as SX-JO), GLOBAL-ORDER (denoted as GO) and SHARED-INLINING (denoted as SI) were implemented with JDK 1.5. We used the Windows version of MONETDB/XQuery 0.12.0 (denoted as MXQ) downloaded from <http://monetdb.cwi.nl/XQuery/Download/index.html>. The experiments were conducted on an Intel Xeon 2GHz machine running on Windows XP with 1GB of RAM. The RDBMS used was Microsoft SQL Server 2005 Developer Edition. Note that we did not study the performance of XML support of SQL Server 2005 as it can only evaluate the first two ordered queries in Figure 5(b).

**Data and query sets.** In our experiments, XBENCH [10] dataset was used for synthetic data. Data-centric (DC) documents were considered with data sizes ranging from 10MB to 1GB. In addition, we used a real dataset, namely DBLP XML [12]. Figure 5(a) shows the characteristics of the datasets used. Two sets of queries were designed to cover different types of ordered XPATH queries. In additional, the cardinality of the results was varied. Figures 5(b) and 5(c) show the benchmark queries on XBENCH and DBLP, respectively. XPATH queries with descendant axes were not included as they had been studied in [6].

**Test methodology.** The XPATH queries were executed in the *reconstruct* mode where not only the non-leaf nodes, but also all their descendants, were selected. Appropriate indexes were constructed for all approaches (except for MONETDB) through a careful analysis on the benchmark queries. Prior to our experiments, we ensured that statistics on relations were collected. The bufferpool of the RDBMS was cleared before each run. Each query was executed 6 times and the results from the first run were always discarded.

| ID  | DC10   |          |          |           |          | DC100    |           |           |           |           | DC1000    |            |           |            |  |
|-----|--------|----------|----------|-----------|----------|----------|-----------|-----------|-----------|-----------|-----------|------------|-----------|------------|--|
|     | MXQ    | SI       | GO       | SX        | SX-JO    | MXQ      | SI        | GO        | SX        | SX-JO     | SI        | GO         | SX        | SX-JO      |  |
| Q1  | 44.17  | 1,042.33 | 843.17   | 58.33     | 58.33    | 80.50    | 5,967.00  | 13,177.17 | 47.67     | 47.67     | 39,152.67 | 85,223.50  | 61.67     | 61.67      |  |
| Q2  | 36.17  | 1,041.17 | 862.33   | 27.67     | 27.67    | 114.67   | 5,967.00  | 7,653.67  | 60.33     | 60.33     | 39,152.67 | 86,271.17  | 44.50     | 44.50      |  |
| Q3  | 492.33 | 4,935.33 | 7,163.00 | 75,236.00 | 5,885.00 | 3,023.67 | 31,229.50 | 43,517.67 | DNF       | 47,664.17 | 64,976.50 | 134,293.83 | DNF       | 368,666.00 |  |
| Q4  | 228.50 | 3,138.83 | 4,517.33 | 2,726.00  | 2,726.00 | 1,364.33 | 17,540.33 | 30,352.50 | 14,266.33 | 14,266.33 | 44,738.67 | 286,369.00 | 56,665.17 | 56,665.17  |  |
| Q5  | 41.83  | 385.33   | 1,359.67 | 13.00     | 28.17    | 85.83    | 1,740.67  | 7,176.50  | 5,133.67  | 209.00    | 7,563.33  | 1,026.17   | 49,795.33 | 1,036.67   |  |
| Q6  | 41.50  | 41.17    | 1,233.67 | 63.67     | 72.83    | 88.67    | 437.67    | 7,121.67  | 339.00    | 248.50    | 1,951.00  | 889.83     | 54,927.67 | 925.67     |  |
| Q7  | 36.33  | 708.67   | 1,594.00 | 63.67     | 78.50    | 81.17    | 4,223.33  | 7,161.33  | 5,236.20  | 208.20    | 30,292.83 | 908.17     | 50,419.83 | 1,000.50   |  |
| Q8  | 39.00  | 688.17   | 1,556.33 | 125.67    | 35.67    | 85.83    | 3,522.17  | 7,301.83  | 365.83    | 222.83    | 6,702.00  | 868.67     | 54,610.83 | 1,144.17   |  |
| Q9  | 36.00  | 91.00    | 3,244.50 | 132.67    | 137.83   | 174.67   | 804.83    | 8,809.00  | 650.83    | 668.67    | 6,264.50  | DNF        | 42,872.00 | 7,992.17   |  |
| Q10 | 39.00  | 72.50    | 5,007.17 | 153.17    | 137.50   | 177.17   | 511.00    | 8,129.83  | 680.17    | 702.33    | 1,720.33  | DNF        | 42,925.17 | 8,456.50   |  |

(a) For DC10, DC100, and DC1000 (in msec)

| ID | MXQ      | SI        | GO        | SX    | SX-JO     | ID | MXQ      | SI        | GO        | SX        | SX-JO    |
|----|----------|-----------|-----------|-------|-----------|----|----------|-----------|-----------|-----------|----------|
| D1 | 1,927.80 | 6,264.17  | 24,975.17 | 55.00 | 55.00     | D3 | 2,143.60 | 82,539.00 | 32,829.17 | 46,827.50 | 2,008.83 |
| D2 | 2,803.00 | 12,596.67 | 39,912.00 | DNF   | 32,605.83 | D4 | 2,859.20 | 81,575.00 | 32,795.00 | 46,820.50 | 1,886.83 |

(b) For DBLP (in msec)

**Fig. 6.** Query Performance (in msec)

## 7.1 Query Evaluation Times

Figures 6(a) (resp. 6(b)) presents the query evaluation times for the approaches on DC (resp. DBLP) dataset. Queries that Did Not Finish within 60 minutes were denoted as DNF.

**Enforcement of Join Order.** The SX and SX-JO columns in Figure 6 describes the effect of enforcing join order in SUCXENT++. Note that we did not enforce the join order for queries Q1, Q2, Q4, and D1 when the PathValue table appears in the translated SQL queries only once.

We made three main observations from our results as follows. First, in almost all cases the query performance improved significantly when join order is enabled. For instance, for DBLP the performance of queries D3 and D4 were improved by factors of 23 and 25, respectively. In fact, 18 out of 24 queries in Figure 6 benefited from join order enforcement. Second, the benefit of this technique increases as the dataset size increases. For instance, for the 1GB dataset the performances of Q5 to Q8 improved by 47 to 59 times. Furthermore, queries that failed to return results previously in 60 minutes (Q3, D2) were now able to return results across all benchmark datasets. Without being privy to optimizer internals, we observed from the query plans of Q3 and Q5–Q8 that the query plan trees consisted of essentially two subtrees. One depicted the plan for computing the V table (lines 03-11 in Figure 4(b)) followed by joining it to the Attribute table (Lines 16-19). The other subtree computed the V table and then returned all the attributes of V (Lines 13-14 in Figure 4(b)). Interestingly, when join order was enforced, the number of joins in the former subtree was reduced and the size of intermediate results were reduced in the later subtree. Consequently, this resulted in a better query plan. For further details on the query plans please refer to [7]. Third, the penalty of join order for most of the benchmark queries, if any, was low on all benchmark datasets. In fact, the largest penalty on the query performance due to join order enforcement was 22ms.

**Comparison with GLOBAL-ORDER and SHARED-INLINING.** Overall SX-JO outperformed both SI and GO in at least 65% of the benchmark queries with the highest observed gain factors being 880 and 1939, respectively. GO showed non-monotonic behavior for Q5–Q8 and as a result the performance of SX-JO was comparable to GO for these queries on DC1000. However, SX-JO significantly outperformed SI for Q5–Q8

(up to 30 times). Note that for DC1000, GO failed to return results for queries Q9 and Q10. Finally, for the DBLP dataset, SX-JO significantly outperformed GO and SI for D1, D3, and D4, with the highest observed gain factor 454 and 114, respectively.

**Comparison with MONETDB.** Our study in the context of MONETDB revealed some interesting results. First, MXQ was 11-164 and 3-74 times faster than GO and SI, respectively, for the majority of the benchmark queries. However, this performance gap was significantly reduced when it was compared against SX-JO. Our results showed that MXQ was 1.3-16 times faster than SX-JO. Surprisingly our approach was faster than MONETDB for 33% of benchmark queries! Specifically, SX-JO was faster than MXQ for Q2, Q5, and Q8 on DC10 and Q1 and Q2 on DC100. Also, for the real dataset (DBLP) SX-JO was faster than MXQ for D1, D3, and D4 with the highest observed factor being 35. Unfortunately, we could not report the results of MXQ for DC1000 because it failed to shred the document. The reason of this problem is that MXQ (Win32 builds) is currently vulnerable to the virtual memory fragmentation in Windows environment. MXQ also does not evaluate predicates applied after reverse axis in reverse document order, but in document order. Therefore, in Q8, it evaluated the second preceding-sibling element in document order, not in reverse document order (not in accordance to W3C XPath recommendation [13]).

## 8 Conclusions and Future Work

In this paper, we presented a scalable storage scheme for ordered XPATH evaluation in relational environment. The mapped SQL queries were forced to execute a “left-to-right” join order. We showed that this technique could improve query performance notably. In addition, our results showed that our proposed approach outperforms other representative *tree-unaware* approaches for the majority of the benchmark queries. Although *tree-aware* approaches were often the best in terms of query performance [1], the “join-order conscious” SUCXENT++ reduced the performance gap between tree-aware and tree-unaware approaches significantly and could outperform a state-of-the-art tree-aware approach (MONETDB) for certain benchmark queries. Importantly, unlike tree-aware approaches, our approach did not require any invasion of the database kernels to improve query performance and could easily be built on top of any off-the-shelf commercial RDBMS. As part of our future work, we are studying the “join order” phenomena encountered during our investigation. We are also exploring other non-invasive mechanisms for improving XPATH query performance on a relational backend.

## References

1. P. BONCZ, T. GRUST, M. VAN KEULEN, S. MANEGOLD, J. RITTINGER, J. TEUBNER. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. *In SIGMOD*, 2006.
2. D. DEHAAN, D. TOMAN, M. P. CONSENS, M. T. OZSU. A Comprehensive XQuery to SQL Translation Using Dynamic Interval Coding. *In SIGMOD*, 2003.
3. D. FLORESCU, D. KOSSMAN. Storing and Querying XML Data using an RDBMS. *IEEE Data Engng. Bulletin*. 22(3), 1999.

4. T. GRUST, J. TEUBNER, M. V. KEULEN. Accelerating XPath Evaluation in Any RDBMS. *In ACM TODS*, 2004.
5. S. PAL, I. CSERI, O. SEELIGER ET AL. XQuery Implementation in a Relational Database System. *In VLDB*, 2005.
6. S. PRAKASH, S. S. BHOWMICK, S. K. MADRIA. Efficient Recursive XML Query Processing Using Relational Databases. *In DKE*, 58(3), 2006.
7. B.-S. SEAH, K. G. WIDJANARKO, S. S. BHOWMICK, B. CHOI, E. LEONARDI. Efficient Support of Ordered XPath Processing in Relational Databases. *Technical Report*, CAIS-05-2006, 2006. Available at <http://www.cais.ntu.edu.sg/~sourav/papers/OrderedXPath-TR.pdf>
8. J. SHANMUGASUNDARAM, K. TUFTE ET AL. Relational Databases for Querying XML Documents: Limitations and Opportunities. *In VLDB*, 1999.
9. I. TATARINOV, S. VIGLAS, K. BEYER, ET AL. Storing and Querying Ordered XML Using a Relational Database System. *In SIGMOD*, 2002.
10. B. YAO, M. TAMER ÖZSU, N. KHANDELWAL. XBench: Benchmark and Performance Testing of XML DBMSs. *In ICDE*, Boston, 2004.
11. C. ZHANG, J. NAUGHTON, D. DEWITT, Q. LUO AND G. LOHMANN. On Supporting Containment Queries in Relational Database Systems. *In SIGMOD*, 2001.
12. DBLP XML Record. <http://dblp.uni-trier.de/xml/>.
13. XML Path Language (XPath) 2.0: W3C Proposed Recommendation 21 November 2006. <http://www.w3.org/TR/xpath20/>

# On Label Stream Partition for Efficient Holistic Twig Join

Bo Chen<sup>1</sup>, Tok Wang Ling<sup>1</sup>, M. Tamer Özsu<sup>2</sup>, and Zhenzhou Zhu<sup>1</sup>

<sup>1</sup> School of Computing, National University of Singapore

{chenbo, lingtw, zhuzhenz}@comp.nus.edu.sg

<sup>2</sup> David R. Cheriton School of Computer Science, University of Waterloo  
tozsuz@uwaterloo.ca

**Abstract.** Label stream partition is a useful technique to reduce the input I/O cost of holistic twig join by pruning useless streams beforehand. The *Prefix Path Stream (PPS)* partition scheme is effective for non-recursive XML documents, but inefficient for deep recursive XML documents due to the high CPU cost of pruning and merging too many streams for some twig pattern queries involving recursive tags. In this paper, we propose a general stream partition scheme called *Recursive Path Stream (RPS)*, to control the total number of streams while providing pruning power. In particular, each recursive path in *RPS* represents a set of prefix paths which can be recursively expanded from the recursive path. We present the algorithms to build *RPS* scheme and prune *RPS* streams for queries. We also discuss the adaptability of *RPS* and provide a framework for performance tuning with general *RPS* based on different application requirements.

## 1 Introduction

An XML document contains hierarchically nested elements, which can be naturally modeled as a labeled ordered tree. Standard query languages for XML usually specify a twig pattern query and retrieve a subset of XML elements in the document. A twig pattern can be represented as a node-labeled tree whose edges are either Parent-Child (P-C) or Ancestor-Descendant (A-D) relationships.

Extensive research efforts have been put into efficient twig pattern query processing with label-based structural joins. Following the early binary structural join algorithms [1][2], Bruno et al. [2] proposed holistic *TwigStack* join algorithm to solve the problem of useless intermediate result in binary structural joins. It produces no useless intermediate result for twig patterns with only A-D relationships, which is defined as optimality. However, *TwigStack* is not optimal for twig query with P-C relationship. Several following works [3][5][6][8][10][9] suggest different ways of optimizing *TwigStack*, such as indexing [6], partitioning [3] label streams, exploring Extended Dewey Label scheme [9], etc.

Most *TwigStack* optimization techniques focus on reducing intermediate results and input I/O cost. [3] further defines the optimality of twig pattern matching as minimal possible I/O cost in reading label streams and maintaining

intermediate results. Though I/O is an important metric in traditional database management, it alone does not well represent the performance in twig pattern query processing, especially with stream partition approach. For example, in [3], the prefix path stream (PPS) partition scheme performs very well in terms of I/O cost. However, its response time is the worst for deep recursive data as a result of high CPU cost of pruning and merging too many streams.

In this paper, in view of the success and limitation of label stream partition in [3], we study the I/O and CPU tradeoffs for stream partition of holistic twig joins and focus on optimizing response time rather than optimizing pure I/O cost addressed previously. In particular,

1. We propose a novel stream partition technique called *recursive path stream (RPS)* partition, which can effectively achieve the I/O benefit of PPS partition [3] while solving PPS's problem of high CPU cost.
2. We also introduce a framework of adaptability of different streaming schemes and further partition of recursive path streams to flexibly fit different application requirements.
3. Our experiment results show that *RPS* is superior to other partition schemes for deep recursive data, while for non-recursive data, *RPS* is better than original *TwigStack* and as good as *PPS*.

Though our discussion in this paper focuses on label stream partition, our technique can be easily combined with other previous works, such as label indexing [6] and Extended Dewey Labeling scheme [9], to utilize their benefits.

The rest of the paper is organized as follows: we present related work in Section 2. In Section 3, we discuss the motivation and our Recursive Path Stream scheme (RPS) in detail. Experiment results are shown in Section 4. Finally, we conclude the paper and discuss possible future research in Section 5.

## 2 Related Work

Twig join processing is central to XML query evaluation. Extensive research efforts have been put into efficient twig pattern query processing with label-based structural joins. Zhang et al. [12] first proposed *multi-predicate merge join (MPMGJN)* based on containment ( $DocId, Start, End, Level$ ) labeling of XML document. The later work by Al-Khalifa et al. [1] proposed an improved stack-based structural join algorithm, called *Stack-Tree-Desc/Anc*. Both of these are binary structural joins and may produce large amount of useless intermediate results. Bruno et al. [2] then proposed a holistic twig join algorithm, called *TwigStack*, to address and solve the problem of useless intermediate results. However, *TwigStack* is only optimal in terms of intermediate results for twig query with only A-D relationship. It has been proven [4] that optimal evaluation of twig patterns with arbitrarily mixed A-D and P-C relationships is not feasible.

There are many subsequent works that optimize *TwigStack* in terms of I/O, or extend *TwigStack* for different problems. In particular, a *List* structure is introduced in *TwigStackList* [8] for wider range of optimality. *TSGeneric* [6] is

based on indexing each stream and skipping labels within one stream. Chen et al. [3] divides one stream (originally associated with each tag) into several sub-streams associated to each prefix path or each tag+level pair and prunes some sub-streams before evaluating the twig pattern. We call this approach as stream partition. Lu et al. [9] uses *Extended Dewey* labeling scheme and scans only the labels of leaf nodes in a twig query. Further techniques of processing twig queries with OR-predicate [5], NOT-predicate [11] and ordered twig queries [10] have also been proposed.

Our proposal is also based on label stream partition like [3]. However, we extend the solution into general optimization of both I/O and CPU cost to reduce response time. It is worth noting that our technique can be easily combined with other works discussed above to achieve their benefits.

### 3 Recursive Path Stream

#### 3.1 Motivation and Terminology

We model XML documents as labeled ordered trees. Each element, attribute and text value in the tree is associated with a label according to some labeling scheme, e.g. *containment* or *prefix* labeling schemes. One XML label uniquely identifies one element in the document. XML queries use twig patterns to match relevant portions of data in an XML document. Twig pattern edges can be parent-child (P-C) or ancestor-descendant (A-D) relationships. XML documents usually have DTD or schema information to specify their structure and to guide users writing queries.

Fig. II(b) shows a sample DTD. Fig. II(c) is a twig pattern query with respect to the DTD in (b). Double lines indicates A-D relationship among query nodes while single line indicating P-C relationship is not shown in the example. A sample XML tree conforming to the DTD is given in Fig. II(a). Elements are associated with containment labels. For illustration purpose, we also show the document order of each element as subscripts  $n$ , and we use  $n$  to refer to the  $n^{th}$  element as well as its label.

To process the query of Fig. II(c) over XML tree in Fig. II(a), original *TwigStack* algorithm [2] scans all the labels of tags  $A$ ,  $B$  and  $C$ . The set of labels of a tag is usually referred to as a *tag stream*, and the process of scanning the tag stream is called *tag streaming*. (We restrict our discussions from stream indexes, though our approach can be easily extended with stream indexes [6].) The tag streams that *TwigStack* algorithm needs to scan for this query are shown in Fig. II(d).

Observe that elements  $A_1$  to  $A_5$  do not contribute to the final results of query  $Q$  in II(c). Therefore, Chen et al. [3] propose to partition each tag stream into *prefix path streams (PPS)* and prune prefix path streams that definitely do not contribute to final results before twig join, thus saving input I/Os. There are 21 prefix paths for sample data in Fig. II(a). Fig. II(e) shows all the streams of paths ending with tag  $A$ . The five prefix path streams of tag  $A$  on the left column can be pruned before processing  $Q$  ([3]) as there are no  $B$  in the prefix path.

Prefix path stream scheme saves input I/Os. However, it needs to check all the paths to prune the useless ones. Moreover, holistic twig join algorithms require

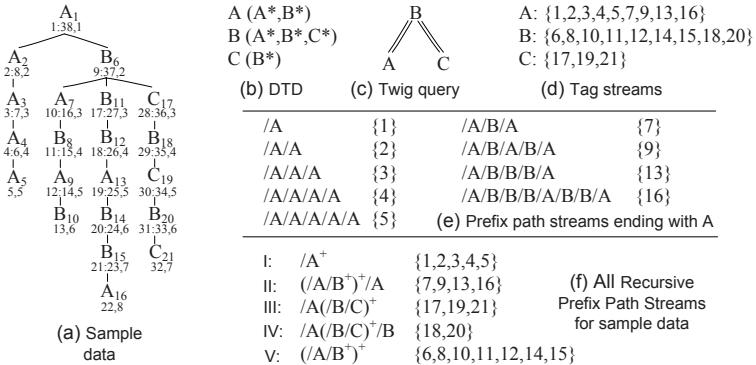


Fig. 1. Example XML document and Query

scanning labels in document order. Therefore, PPS scheme has to merge-sort all the prefix path streams for each tag during run time. The pruning and merge-sorting can be CPU expensive for deep recursive data with many prefix paths for each tag. In Fig. II(e), we first need to prune 5 streams, then merge-sort 4 streams on the right column during holistic twig join.

We observe that prefix paths for  $A$  in Fig. II(e) can be grouped and represented as the first two special paths in Fig. II(f), where the ‘+’ sign in  $/A^+$  indicates there may be one or more consecutive  $/A$ 's in a prefix path. We term the special path as *Recursive Path*. The following introduces the terminology used in the paper.

**Recursive Path (RP)** is a special representation of a set of prefix paths that are recursively built on some tags. One or a sequence of tags in RP enclosed within ‘+’ can be recursively expanded to represent prefix paths of different lengths. We call tags enclosed within a ‘+’ as a **Recursive Component (RC)**. RC's can be recursive, e.g.  $(/A/B^+)^*/A$ . Only P-C relationship is allowed between consecutive tags in RP. Each RP has a set of RC's. We can also view one prefix path as an RP with empty RC set, representing a singular path set of itself. If two RP's has the same tag sequence, but different RC sets, they can be combined into a **general form** such that the RC set of the general form is the union of RC sets of the two RP's. Each RP is associated with a label stream, called **Recursive Path Stream (RPS)**. This stream contains the labels of elements of all the prefix paths represented by the RP in document order.

In Fig. II(f), we have only five recursive paths for 21 prefix paths. For query node  $A$  in Fig. II(c), we can prune RP I and only scan the stream of II since there is no  $B$  in I. In this way, we save both I/O and CPU cost. We call RP II and its stream as the **Potential Solution Path (PSP)** and **Potential Solution Stream** for the query node  $A$ .

### 3.2 Building RPS Scheme from XML Data

We present the algorithm to extract RPS from XML data in Fig. 2. The algorithm, *BuildRPS*, uses SAX event parser and extracts recursive paths and their

**Algorithm 1.** BuildRPS

---

**Input:** Events  $e$  from SAX parser;  
**Output:**  $RPS$ ; /\*  $RPS$  maps RP to stream \*/

1. initialize Stack  $ST$ ; /\*  $ST$  is the stack for start tags \*/
2. initialize empty Hashtable  $RPS$ ;
3. **while** there are more events  $e$
4.   **if**  $e$  is start tag **then**
5.     push tag  $t$  of  $e$  onto  $ST$ ;
6.     scan from the bottom to top of  $ST$  to get path  $p$  for the element;
7.     let  $len$  = the number of tags in  $p$ ;
8.     **for** ( $n = 1, n \leq \lfloor len/2 \rfloor, n++$ )
9.       **while** (there are consecutive occurrences of a same sequence of tags of length  $n$  in  $p$ )
10.       /\* checking from root to leaf to ensure same PP gives same RP \*/
11.       change  $p$  by replacing all occurrences of the same sequence by one recursive component in  $p$ ;
12.       let  $len$  = new number of tags in  $p$ ; /\*  $len$  should be decreased \*/
13.     **end while**
14.   **end for**
15.   **if** (there is a path  $p'$  in  $RPS$  with the same tag sequence of  $p$ ) **then**
16.     generate the general form  $p''$  of  $p'$  and  $p$ ;
17.     /\* the recursive component set of  $p''$  is the union of  $p'$  and  $p$ 's RC set \*/
18.     associate  $p''$  with the stream of  $p'$  and remove  $p'$  in  $RPS$ ;
19.   **else**   put  $p$  into  $RPS$ ;
20.   generate and append the *start* and *level* values of current element's label to corresponding recursive path;
21.   **else if**  $e$  is end tag **then**
22.     pop  $ST$ ;
23.     complete the label of  $e$  in  $RPS$  by generating and adding the *end* value;

**end while**

---

**Fig. 2.** Algorithm for building Recursive Path Stream (RPS) scheme

label streams with one pass of the data. This version of *BuildRPS* only handles XML elements, but can be easily extended for attributes.

*BuildRPS* works in three steps for each element in the XML document. Step 1 (lines 4–14) computes the element's path  $p$  and compacts it into recursive path (RP). It searches for consecutive occurrences of the same tag sequences of length  $n$  (where  $n$  ranges from 1 to half of the length of  $p$  since the length of the tag sequence can be at most the half of  $p$  in order to have two consecutive occurrences of the same tag sequences) from root to leaf of  $p$ . If there are such consecutive occurrences, lines 11 & 12 compact  $p$  by replacing the multiple same sequences by one sequence as the recursive component (RC) and set length  $len$  to the new length of  $p$ . Step 2 (line 15–20) combines RPs of the same tag sequence into their *general form* and appends the partial label of *start* and *level* values to the corresponding stream. This is to ensure that two different RPs produced by the algorithm represents two disjoint set of prefix paths. Step 3 (lines 21–23) completes the label of the ending element by adding the *end* value.

*Example 1.* Consider how *BuildRPS* algorithm extracts the RP  $(/A/B^+)^+/A$  and its label stream in Fig. II(f) for data of Fig. II(a). When the scan reaches start tag of  $A_7$ , steps 1 first computes its path  $p_1$ ,  $/A/B/A$ . Since  $p_1$  is uncompactable, and this is the first path with tag sequence  $(A, B, A)$ , step 2 associates  $p_1$  with the partial label of  $A_7$  without *end* value. Then after start tag of  $A_9$  is reached, the algorithm gets the path  $p_2$ ,  $/A/B/A/B/A$ , compacts it into recursive path  $rp_1$ ,  $(/A/B)^+/A$ , with  $\{( /A/B )^+\}$  as the RC set. Now, since  $p_1$  and  $rp_1$  have the same tag sequence and their general form is identical to  $rp_1$  with a singular RC set, step 2 replaces  $p_1$  by  $rp_1$  and appends the partial label of  $A_9$  to the label stream. Then after scanning the end tags of  $A_9$  and  $A_7$ , step 3 completes their labels with *end* values. When the scan reaches the start tag of  $A_{13}$ , step 1 computes  $rp_2$ ,  $(/A/B^+)/A$ , then step 2 combines it with  $rp_1$  to get the general form  $(/A/B^+)^+/A$  to replace  $rp_1$  and appends the stream. Similar actions are taken when the start and end tags of  $A_{16}$  and end tag of  $A_{13}$  are reached.

Note that step 2 does not produce  $/A(/B/A)^+$  for  $p_2$ ,  $/A/B/A/B/A$ , since it searches from the root to leaf. The algorithm first finds the consecutive occurrences of  $/A/B$  and immediately changes  $p_2$  into  $rp_1$  which does not contain consecutive occurrences of  $/B/A$  any more.

The time complexity of *BuildRPS* is  $O(D * L^3)$ , where  $D$  and  $L$  are the size and maximum depth of the document. The followings are two properties of RPS scheme computed by *BuildRPS*. The proofs are omitted due to lack of space.

**Property 1:** Same prefix paths are always compacted to the same recursive path with shortest possible tag sequence.

**Property 2:** Two different recursive paths represent two disjoint prefix path sets as well as disjoint label streams.

### 3.3 Identifying Potential Solution Paths

We now discuss the process of identifying potential solution (and pruning useless) paths for a twig pattern query. The algorithm is based on the following two properties of a recursive path.

**Property 3:** For any two tags  $T_1$  and  $T_2$  in a recursive path  $P$ ,  $T_1$  is an **ancestor tag** of  $T_2$  if  $T_1$  appears before  $T_2$  in  $P$  or there exists some recursive component in  $P$  containing both  $T_1$  and  $T_2$

**Property 4:** For any two tags  $T_1$  and  $T_2$  in a recursive path  $P$ ,  $T_1$  is a **parent tag** of  $T_2$  if  $T_1$  appears before  $T_2$  consecutively in  $P$  or there exists some recursive component  $RC$  in  $P$  such that  $T_2$  is the first tag and  $T_1$  is the last tag of  $RC$ .

*Example 2.* Consider the recursive path  $/A(/B/C/D)^+$ .  $A, C$  and  $D$  are all ancestor tags of  $B$  since 1)  $A$  appears before  $B$  and 2) there is one recursive component containing all  $B, C$  and  $D$ .  $C$  and  $D$  will appear before  $B$  if we expand  $(/B/C/D)^+$  once to get  $/A/B/C/D/B/C/D$ . However, only  $A$  and  $D$  are the parent tags of  $B$  as they appear before  $B$  consecutively after the expansion.

**Algorithm 2.** IdentifyPSP

---

**Input:** Twig query  $Q$  and RPS partition scheme  $P$   
**Output:** Potential Solution Path sets  $Psets$  for all query node  $N$  in  $Q$

1. initialize  $Pset$  of each query node as empty set.
2. **depth first search** query twig  $Q$ , upon returning from current query node  $N$ ;
3. let  $Cset$  of  $N$  be an empty set /\*  $Cset$  is “Candidate PSP set” \*/
4. get query path  $qp$  from query root to  $N$
5. **if**  $N$  is leaf query node **then**
6.     let  $Cset$  be all recursive paths ending with tag  $N$  in  $P$ ;
7. **else if**  $N$  is non-branching internal query node **then**
8.     let  $Cset = \text{getCset}(N, PSet \text{ of child of } N)$ ;
9. **else if**  $N$  is branching query node **then**
10.    **for**  $Pset$  of each child  $Ci$  of  $N$ 's children
11.     let  $Cset_i = \text{getCset}(N, Pset)$ ;
12.    **end for**
13.     let  $Cset$  be the intersection of all  $Cset_i$ 's;
14.    **for** each  $rp$  in  $Cset$
15.     **if**  $\text{checkPSP}(rp, qp) == \text{true}$  **then** put  $rp$  in  $Pset$  of  $N$ ;
16.    **end for**
17. **end depth first search**
18. **for** each query node  $N$
19.    **for** each  $rp$  in  $Pset$  of  $N$
20.     **if**  $\neg\exists rp'$  in  $Pset$  of root s.t. tag sequence of  $rp'$  is a prefix of  $rp$  **then**
21.       remove  $rp$  from  $Pset$  of  $N$ ;
22.     **end for**
23. **end for**

Function  $\text{getCset}(N, childPset)$  /\* get  $Cset$  of  $N$  based on  $Pset$  of  $N$ 's child \*/

1. let  $Cset$  be empty set;
2. **for** each  $rp$  in  $childPset$
3.     put each RP whose tag sequence is a prefix of  $rp$  and ends with  $N$  into  $Cset$ ;
4.     **for** each recursive component  $rc$  containing but not ending with  $N$  in  $rp$
5.       get tag sequence  $ts$  by repeating tags up to  $N$  in  $rc$  once;
6.       put into  $Cset$  the RP of tag sequence from the root to repeated  $N$  in  $ts$ ;
7.     **end for**
8. **end for**
9. return  $Cset$ ;

Function  $\text{checkPSP}(rp, qp)$  /\* check if  $rp$  is potential solution path of  $qp$  \*/

1. let tag set  $s_1$  be  $N$  where  $N$  is the leaf node of  $rp$ ;  
/\*elements in tag set are of the same name, differentiated by the positions in  $rp$ \*/
2. **for** each  $qp$  tag  $T$  from leaf to root /\*  $T$ 's parent is dummy if  $T$  is root \*/
3.     let  $PT$  be parent tag of  $T$  in  $qp$  and  $E$  be the edge between  $PT$  and  $T$ ;
4.     **if**  $T$  is the root **then** return BOOLEAN( $E$  is A-D OR  $s_1$  contains root of  $rp$ );
5.     let tag set  $s_2$  be  $\{e_2 \mid e_2$  and  $pt$  have identical tag  $\wedge$   
 $\exists e_1 \in s_1$  s.t.  $e_2$  is the parent (or ancestor based on  $E$ ) tag of  $e_1$  in  $rp\}$ ;
6.     **if**  $s_2$  is empty **then** return false;
7.     **else**       let  $s_1$  be  $s_2$ ;
8. **end for**

**Fig. 3.** Algorithm for Identifying Potential Solution Paths in RPS scheme

We show algorithm *IdentifyPSP* in Fig. 3. It identifies the Potential Solution Path (PSP) set ( $Pset$ ) for each query node in a given twig query, with two phases: bottom-up pruning from query leaves and top-down pruning from query root. The bottom-up phase first propagates branching information for pruning from branches to the branching nodes; whereas top-down phase then propagates the combined branching information to each individual branch.

In the bottom-up pruning phase (lines 2-17 of Main), it visits each query node  $N$  in depth first order. Upon returning from  $N$ , it first computes  $N$ 's Candidate Potential Solution Path set ( $Cset$ ) (lines 3-13), then checks each recursive path  $rp$  in  $Cset$  if it is a PSP to be put into  $Pset$  (lines 14-16). Note that for branching node, the  $Cset$  is the intersection of the  $Csets$  computed based on the  $Psets$  of each child query node. We will shortly discuss how to compute  $Cset$  of a query node based on its child's  $Pset$ . In the top-down pruning phase (lines 18-23), for  $Pset$  of each non-root query node, it removes all the recursive paths (RP) for which there exists no RP as its prefix in the  $Pset$  of the query root.

There are two auxiliary functions for the algorithm: *getCset* and *checkPSP*. Function *getCset* finds the  $Cset$  of query node  $N$  based on the  $Pset$  of  $N$ 's child. It puts into  $Cset$  all recursive path  $rp$  such that  $rp$  ends with  $N$  and is a prefix of either 1) any  $rp'$  in  $Pset$  of  $N$ 's child or 2) tag sequence expanded from  $rp'$  by repeating any single recursive component once. Function *checkPSP* checks if the given recursive path  $rp$  is the PSP for query path  $qp$ . It recursively scans query tag  $T$  from the leaf to the root of  $qp$ . For each  $T$  and  $T$ 's parent query node  $PT$ , it computes the  $T$ 's ancestor (or parent depending on the query edge between  $T$  and  $PT$ ) tags that are same to tag name  $PT$ . When none can be found or  $T$  is the root query node, *checkPSP* returns.

The time complexity of *IdentifyPSP* is  $O(|Q| * |rp| * (F_Q * |rc| + D_q * D_{rp}))$ , where  $|Q|$ ,  $|rp|$ ,  $F_Q$ ,  $|rc|$ ,  $D_q$  and  $D_{rp}$  are number of query nodes, number of RPs (for each tag if we have a mapping from tags to their RPs), maximum query fan-out, maximum number of RCs in one RP, query depth and maximum depth of RP respectively. Since most of the above values are usually small, saving in IO is usually worth the efforts in pruning. The following theorem shows the correctness of *IdentifyPSP*. However, due to lack of space, we cannot provide the proofs for the time complexity and the theorem.

**Theorem 1.** *Given query  $Q$  and RPS scheme, labels of streams pruned by *IdentifyPSP* algorithm do not contribute to final answer of  $Q$ .*

*Example 3.* Let us trace algorithm *IdentifyPSP* on the query and RPS scheme in Fig. 1(c) and (f). In bottom-up phase, depth first search first returns from query node  $A$ . Within the two candidate RPs {I, II} of  $A$ , only RP II in Fig. 1(f) is identified as PSP since query root  $B$  appears as an ancestor tag of  $A$  in II, but not in I. So, the current PSP set of  $A$  is {II}. Similarly, {III} is the PSP set for  $C$ . Now, according to function *getCset*, the candidate PSP sets for  $B$  are {V} based on  $A$ 's PSP set and {IV, V} based on  $C$ 's PSP set (IV is in the candidate PSP set of  $B$  since it is a prefix of the expansion of RC in III). So the intersection is {V}, which is then identified as the PSP set of  $B$ . The top-down pruning does not take effects in this case. Thus the final PSP set of

$A$ ,  $B$  and  $C$  are {II}, {V} and {III} respectively. However, suppose we modified the sample data to have one more RP as  $/A/D^+/B/A$ , it would be PSP for  $A$  after bottom-up phase but pruned after top-down phase since  $V$  is not its prefix.

### 3.4 Adaptability of Different Stream Partition Schemes

As mentioned in Sec. 3.1, uncompactable prefix path is a special case of recursive path. The RPS scheme is a generalization of PPS. Applying RPS to non-recursive data generates the same stream partition as PPS. Therefore, it is safe to replace PPS with RPS in non-recursive data. Besides, we can further partition streams in RPS according to different application requirements. For example, when there are many  $A$ 's at depths more than three, but most queries are only interested in  $A$ 's at depth less than or equal to two, we can partition the stream associated with  $/A^+$  into streams  $/A^{1:2}$  and  $/A^{3+}$ , meaning  $RC/A$  can be repeated at most twice and at least three times respectively.

However, for irregular data, even RPS may generate too many streams and result in long query response time. For example, if we change the DTD in Fig. 1(a) as “every element of  $(A, B, C)$  can have any element of the three as their children”, we may have the following deep uncompactable data path

$$/A/B/C/B/A/B/C/A/C/B/A/B/C/\dots$$

In such case, it is better to use non-partitioned Tag Streaming to avoid merge-sorting overwhelming number of streams during holistic twig joins.

## 4 Experimental Evaluation

We experimentally compare the performance of RPS with non-partitioned tag stream scheme and existing partition schemes: PPS and Tag+level. Results show that, RPS and PPS are comparable and better than Tag or Tag+level in non-recursive or light recursive data (e.g. XMark). In deep recursive data (e.g. TreeBank), RPS significantly out-performs others for total query response time.

### 4.1 Experimental Settings

**Implementation and Hardware.** We implemented all algorithms in Java. Different stream partition schemes were compared based on TwigStack holistic join [2]. The experiments were performed on a normal PC with 2.6GHz Pentium 4 processor and 1GB RAM running Windows XP.

**XML Data Sets.** We use two well-known data sets (XMark and TreeBank) for our experiments. The characteristics and the number of streams for each partition technique of these two data sets are shown in Table 1. We choose these two data sets because XMark is light recursive with non-recursive tags, while TreeBank is deep recursive. In this way, we can study the performance of various stream partition methods with different levels of recursion in XML data.

**Table 1.** XML Data Sets

|            | XMark       | Treebank    |
|------------|-------------|-------------|
| Size       | 113MB       | 82MB        |
| Nodes      | 2.0 million | 2.4 million |
| Max Depth  | 12          | 36          |
| Ave Depth  | 5           | 8           |
| Tags       | 75          | 251         |
| Tag+Level# | 119         | 2237        |
| PPSs #     | 514         | 338724      |
| RPSs #     | 415         | 119748      |

**Table 2.** Tested Queries

|     |                                          |
|-----|------------------------------------------|
| XM1 | //site/people/person/name                |
| XM2 | //site/people/person[/name]//age         |
| XM3 | //text[//bold]//emph//keyword            |
| XM4 | //text[emph/keyword]/bold                |
| XM5 | //listitem[//bold]/text[//emph]//keyword |
| TB1 | //S//ADJ//MD                             |
| TB2 | //VP//DT//PRP_DOLLAR_                    |
| TB3 | //PP//NP/VBN//IN                         |
| TB4 | /S//VP//PP//NP/VBN//IN                   |
| TB5 | //S//NP//PP/TO[//VP_-NONE_]//JJ          |

**Queries.** We select a wide range of representative queries (shown in Table 2) for each data set (XM for XMark and TB for TreeBank). In particular, XM1 and XM2 contain non-recursive tags, while the rest all contain recursive tags. XM1 is a path query. XM2–4, TB1–4 are simple twig queries with only one branching node of fan-out two. Except incoming root query edge, XM3 and TB1 have only A-D edges; XM4 and TB3 have only P-C edges; while XM2, TB2 and TB4 have a mixture of A-D and P-C edges. For complex twig queries, XM5 has two branching nodes whereas TB5 has one branching node of fan-out three. The number of various label streams for all the tags of each query before and after pruning is shown in Table 3. We can see the number of RPSs is much smaller (up to 67% less) than PPSs.

**Performance Measures.** We compare RPS with non-partitioned Tag streams and existing PPS and tag+level partition schemes. The presented performance measures include pruning time, IO time of reading labels, CPU time of structural join (including merge-sorting streams) and total response time of each query. The IO time and CPU time of joins are estimated by reading all labels into memory (IO time) before in-memory structural join (CPU time). Although the number of labels (or bytes) scanned for each query is also an important measure for the

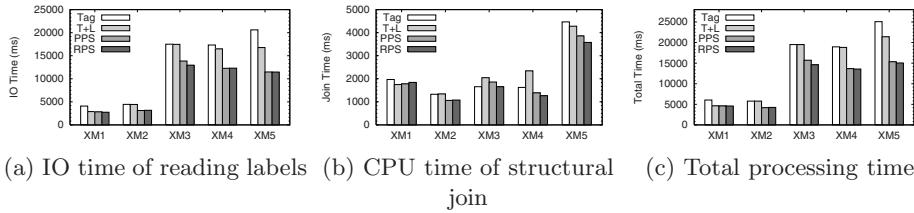
**Table 3.** Number of Streams before and After Pruning for various Partition Schemes

|     | Tag + Level |       | PPS    |       | RPS    |       |
|-----|-------------|-------|--------|-------|--------|-------|
|     | before      | after | before | after | before | after |
| XM1 | 7           | 4     | 11     | 4     | 11     | 4     |
| XM2 | 8           | 6     | 12     | 5     | 12     | 5     |
| XM3 | 27          | 25    | 330    | 198   | 240    | 144   |
| XM4 | 27          | 25    | 330    | 132   | 240    | 96    |
| XM5 | 31          | 23    | 348    | 198   | 249    | 99    |
| TB1 | 62          | 46    | 12561  | 1623  | 5126   | 743   |
| TB2 | 87          | 86    | 38527  | 2455  | 12067  | 814   |
| TB3 | 118         | 100   | 97285  | 1164  | 29563  | 624   |
| TB4 | 177         | 138   | 123669 | 1874  | 38693  | 798   |
| TB5 | 209         | 182   | 132503 | 2805  | 42915  | 1341  |

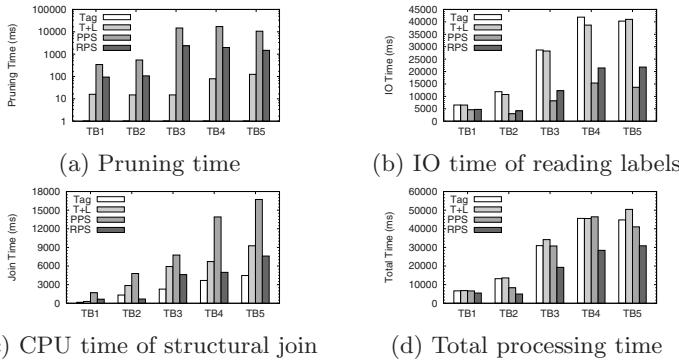
effectiveness of partition schemes, it is not shown due to space limitations as their experiment results are similar to IO time.

## 4.2 Experiment Results and Analysis

We show the experiments results for XMark data in Fig. 4. We did not show the pruning time for XMark as it is only a few milliseconds, for all queries, thus is a negligible component of total response time. It is clear that holistic twig join with RPS partition is faster than Tag+Level (T+L) partition and non-partitioned Tag in both input reading and structural join as a result of less labels scanned and processed. We can also observe that RPS is comparable to PPS for XM1 and XM2 containing non-recursive query tags and slightly better than PPS for XM3–5 containing recursive query tags in terms of structural join and total response time. Theoretically, the number of labels scanned in PPS is less than or equal to RPS. So, it is interesting to see RPS is better than PPS in input reading for XM3 as shown in Fig. 4(a). This is the result of the larger overhead of PPS to read the same number of labels in more streams compared to RPS.



**Fig. 4.** Experimental Results for XMark dataset (metrics of different scales)



**Fig. 5.** Experimental Results for TreeBank dataset (metrics of different scales)

The results for TreeBank data set are shown in Fig. 5. We can see from Fig. 5(a), RPS is much faster than PPS, but slower than Tag+Level in pruning phase as expected. In reading inputs (Fig. 5(b)), PPS is the best since it reads the least amount of labels by pruning more label streams; RPS is a bit slower than PPS, but much faster than Tag and Tag+level. For CPU time of structural join

(Fig. 5(c)), non-partitioned Tag scheme is the best. Although PPS processes the least amount of labels, it is still the worst in structural join time due to high cost of merge-sorting too many streams. RPS is better than Tag+level in structural join time in general because RPS processes much less labels, which outweighs the overhead of merge-sorting more streams. For RPS alone, although it is not the best in any of the pruning, input reading or structural join, the beneficial trade-off between IO and CPU helps RPS to be the best in overall query response time (up to 2 times faster than the most competitive ones) as shown in Fig. 5(d).

## 5 Conclusion and Future Work

In this paper, we propose a novel stream partition scheme for efficient holistic twig joins, namely recursive path stream. RPS scheme is a generalization of prefix path stream proposed in [3]. Experiment results show that RPS is more efficient than other stream partition techniques in recursive XML data while it is as good as PPS and better than others in non-recursive data. As a part of future work, we would like to study the cost model for holistic twig joins with stream partition and indexing.

## References

1. S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. Structural joins: A primitive for efficient XML query pattern matching. In *Proc. of ICDE Conference*, pages 141–152, 2002.
2. N. Bruno, D. Srivastava, and N. Koudas. Holistic twig joins: optimal XML pattern matching. In *Proc. of SIGMOD Conference*, pages 310–321, 2002.
3. T. Chen, J. Lu, and T. W. Ling. On boosting holism in XML twig pattern matching using structural indexing techniques. In *Proc. of SIGMOD Conference*, 2005.
4. B. Choi, M. Mahoui, and D. Wood. On the optimality of the holistic twig join algorithms. In *Proc. of DEXA*, pages 28–37, 2003.
5. H. Jiang, H. Lu, and W. Wang. Efficient processing of XML twig queries with or-predicates. In *Proc. of SIGMOD Conference*, 2004.
6. H. Jiang, W. Wang, H. Lu, and J. Yu. Holistic twig joins on indexed XML documents. In *Proc. of VLDB Conference*, pages 273–284, 2003.
7. C. Li, T. W. Ling, and M. Hu. Efficient processing of updates in dynamic XML data. In *Proc. of ICDE*, 2006.
8. J. Lu, T. Chen, and T. W. Ling. Efficient processing of XML twig patterns with parent child edges: a look-ahead approach. In *Proc. of CIKM*, pages 533–542, 2004.
9. J. Lu, T. W. Ling, C. Chan, and T. Chen. From region encoding to extended dewey: On efficient processing of XML twig pattern matching. In *Proc. of VLDB Conference*, pages 193–204, 2005.
10. J. Lu, T. W. Ling, T. Yu, C. Li, and W. Ni. Efficient processing of ordered XML twig pattern. In *Proc. of DEXA*, 2005.
11. T. Yu, T. W. Ling, and J. Lu. Twigstacklistnot: A holistic twig join algorithm for twig query with not-predicates on XML data. In *Proc. of DASFAA*, 2006.
12. C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. On supporting containment queries in relational database management systems. In *Proc. of SIGMOD Conference*, pages 425–436, 2001.

# Efficient XML Query Processing in RDBMS Using GUI-Driven Prefetching in a Single-User Environment

Sandeep Prakash<sup>1</sup>, Sourav S. Bhowmick<sup>1,2</sup>, Klarinda G. Widjanarko<sup>1,2</sup>,  
and C. Forbes Dewey Jr.<sup>3</sup>

<sup>1</sup> School of Computer Engineering, Nanyang Technological University, Singapore

<sup>2</sup> Singapore-MIT Alliance, Nanyang Technological University, Singapore

<sup>3</sup> Division of Biological Engineering, Massachusetts Institute of Technology, USA  
`{assourav, klarinda}@ntu.edu.sg, cfdevey@mit.edu`

**Abstract.** In this paper, we address the problem of efficient processing of XQueries in single-user relational environment where the queries are formulated using a user-friendly GUI. We take a novel and non-traditional approach to improving query performance by *prefetching data during the formulation of a query*. The latency offered by GUI-based query formulation is utilized to prefetch portions of the query results. To realize this, we present an algorithm for prefetching based on data synapses statistics and GUI actions during visual query formulation. Experimental evaluation indicates that prefetching is viable as the combined time taken by all the prefetching operations is not significantly more than normal query execution time. Our experiments in the context of biological data show that prefetching improves the query response time by 7-96% with a greater improvement for larger data sets. Also, we show the impact of errors committed by users during query formulation on the query performance.

## 1 Introduction

Querying XML data involves two key steps: *query formulation* and *efficient processing* of the formulated query. However, due to the nature of XML data, formulating an XML query using an XML query language such as XQuery requires considerable effort. A user must be completely familiar with the syntax of the query language, and must be able to express his/her needs accurately in a syntactically correct form. In many real life applications (such as life sciences) it is not realistic to assume that users are proficient in expressing such textual queries. Hence, there is a need for a user-friendly visual querying schemes to replace data retrieval aspects of XQuery.

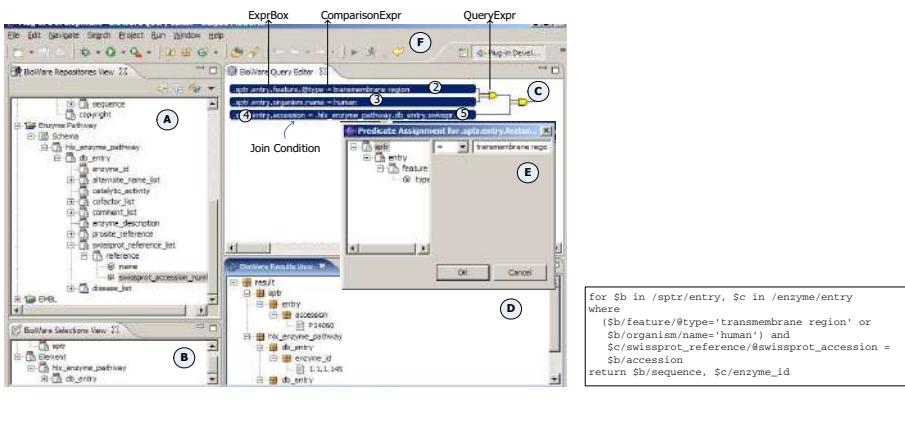
In this paper, we address the problem of efficient processing of XQueries in the relational environment where the queries are formulated using a user-friendly GUI. The work presented here is part of our ongoing research of building a system called *Da Vinci's Notebook* that would empower biologists to explore huge volumes of experimental biology data. We take a novel and non-traditional approach to improving query performance by *prefetching data during the formulation of a query in a single-user environment*. The latency offered by the GUI-based query formulation is utilized to prefetch portions of the query results. In order to expedite XML query processing

using such GUI-based prefetching two key tasks must be addressed. First, given a user-friendly visual query interface, GUI actions that can be used as indicators to perform prefetching need to be identified. Second, each GUI action can possibly lead to more than one prefetching operation. Therefore, an algorithm needs to be designed to select the “best” operation. In this paper, we address these issues in detail. A short overview of this approach appeared as a poster paper in [3].

To the best of our knowledge, this is the first work that makes a strong connection between prefetching-based XML query processing and GUI-based query formulation. The key advantages of our approach are as follows. First, our optimization technique is built *outside* the relational optimizer and is orthogonal to any other existing optimization techniques. Hence, our approach provides us with the flexibility to “plug” it on top of any existing optimization technique for processing XML data in relational environment. Second, our approach is not restricted by the underlying schema of the database. As a result, it can easily be integrated with any relational storage approaches. Third, the prefetching-based query processing is transparent from the user. Consequently, there does not exist any additional cognitive overhead to the users while they formulate their queries using the GUI. Finally, our non-traditional approach noticeably improve the performance of XML query execution. As we shall see in Section 5 our experiments with biological data indicate a performance improvement of 7% to 96% with an increasing improvement as the size of the data grows. Moreover, we also show that errors committed by users while formulating queries *do not* significantly affect the query performance.

## 2 Visual Query Interface

In this section, we present the visual interface which we shall use in the rest of the paper for formulating XML queries. Ideally, a full implementation of the GUI-driven prefetching system would require a fully-functional XQuery support. However, it is also true that a visual interface is useful when it serves the needs of the majority of the users



**Fig. 1.** Visual query interface and XQuery representation

in expressing majority of their queries, which are typically simple [1]. A complete but too complex graphical interface would fail both in replacing the textual language and in addressing all the users' needs [1]. Furthermore, the focus of this paper is to study the effect of GUI-driven prefetching on XML query processing and not design of a complete visual interface for formulating XML queries. Hence, we implemented an interface that supports simpler types of XQuery. These queries are sufficient to justify the positive contributions made by the GUI-based prefetching technique. Specifically, the syntax of the basic XQuery query that can be formulated using our GUI is as follows. Note that we assume that the DTDs/XML schemas of data sources are available to the user during query formulation.

```

FOR $x_1 \text{ in } p_1, \dots, x_n \text{ in } p_n$
WHERE W
RETURN r_1, r_2, \dots, r_k

```

where  $p_i$  is a simple linear path expression,  $W$  is a set of predicates that are connected by AND/OR operator(s). A predicate  $w \in W$  can be one of the two forms:  $s_i \text{ op } c$  or  $s_i \text{ op } s_j$  where  $s_i$  and  $s_j$  are path expressions that may contain a selection predicates and  $c$  is a constant. The variable  $r_i$  is a simple path expression.

Our system allows the user to formulate visual queries in an intuitive manner without having to learn any query language. The user interface (Figure 1(a)) is presented as an adjustable multi-panel window comprising the following items. The *Repositories View* (labelled A) occupies the left pane. It serves as a data source browser in which the user can view the list of available data sources and their respective structures in terms of a tree display of their DTD/XML Schema. Showing multiple data sources allows the formulation of queries spanning more than one source. The data sources shown in Figure 1(a) are SWISSPROT and ENZYME.

The *Query Editors* are stacked in the middle pane (labeled C), with tabs for navigating between queries. It enables the user to specify the WHERE clause. The user drags the node to be queried from the *Repositories View* and drops it in a *Query Editor*. A *Condition Dialog* (labeled E), appears and the user is expected to fill in the condition that should be satisfied by the selected node. In Figure 1(a), the selected node is `/sptr/entry/feature/@type` and the condition is "`=transmembrane region`" thus forming the predicate `.sptr.entry.feature.@type= "transmembrane region"` (labeled 2). This expression is called Comparison Expr and the visual representation of a ComparisonExpr type is referred to as ExprBox.

The user can combine two or more visual components that represent the ComparisonExpr by dragging a region around them and assigning an AND or OR condition. In Figure 1(a), the first two ComparisonExpr (labeled 2 and 3) are combined using the OR operator thus forming the QueryExpr `(.sptr.entry.feature.@type= "transmembrane region" OR .sptr.entry.organism.name= "human")`. In order to specify a join condition two nodes, each representing one side of the join condition are selected and dragged on to the *Query Editor*. This is shown by the labels 4 and 5 in Figure 1(a). The visual representation of a QueryExpr type is also referred to as ExprBox.

The *Selections View* (labeled B) is a drop target for nodes dragged from the *Repositories View* and displays the nodes that will be visible in the result of the query. This enables the visual formulation and representation of the XQuery RETURN clause. The user can execute the query by clicking on the “Run” icon in the *Query Toolbar*. The *Results View* (labeled D) displays the query results.

To formulate a query, the user first selects the nodes that should be present in the RETURN clause. For instance, in Figure 1(a), the nodes selected are `sequence` and `enzyme_id` indicating that the user only wants to view these elements in the result. Next, the predicates in the WHERE clause are formulated in the *Query Editor*. The visual constructs in the *Query Editor* and *Selections View* need to be translated to formulate a complete XQuery. Each `ComparisonExpr` or `QueryExpr` can be combined to obtain a `Query` type. The translation to XQuery can be easily done by following the syntax presented earlier. Figure 1(b) shows the XQuery corresponding to Figure 1(a).

### 3 Computing Query Formulation Time

Our query processing approach utilizes the user’s query formulation time to prefetch results of the intermediate queries. To determine the time available for prefetching (and to measure the improvement provided by prefetching), the time required to formulate a query visually needs to be measured. This is referred to as the *query formulation time* (*QFT*). It is the duration between the time the first predicate is added and the execution of the “Run” command as prefetching can start only when the first predicate is known.

We have used the Keystroke-Level Model (KLM) [4] to calculate QFT. The KLM is a simple but accurate means to produce quantitative, *a priori* predictions of task execution time. These times have been estimated from experimental data [4]. The basic idea of KLM is to list the sequence of keystroke-level actions that the user must perform to accomplish a task, and sum the time required by each action. The KLM has been applied to many different tasks such as text editing, spreadsheets, graphics applications, handheld devices, and highly interactive tasks [4][5].

Figure 2(a) lists average task times for a subset of *physical operators* (K (key-stroking), P (pointing), H (homing), and D (drawing)) as defined by KLM [4]. Figure 2(b) depicts the estimated times for a set of *atomic actions* for visual query formulation. Note that the times are computed using the physical operators in Figure 2(a). Figure 2(c) shows the list of tasks the user needs to perform in order to formulate a query. Each task consists of a set of atomic actions (Figure 2(b)). For example, adding a join predicate (Task T2) involves selecting the two join nodes (Action A1 twice) and dragging them on to the *Query Editor* (Action A2). The estimated time taken to perform each task is simply the sum of average times of the atomic actions.

Note that *QFT* does not include higher level mental tasks for formulating a query such as planning a query formulation strategy. These tasks depend on what cognitive processes are involved, and is highly variable from situation to situation or person to person. We assume that the user has already planned the set of actions he/she is going to take to formulate his/her query and any other mental tasks. That is, our *QFT* in the following discussion consists of a sequence of physical operators only. This assumption enables us to investigate the impact of prefetching for *minimum QFT* for a particular

| Notation | Physical Operator                             | Average Time (s) |
|----------|-----------------------------------------------|------------------|
| K        | Keystroke                                     | 0.28             |
| T(n)     | Type a sequence of n characters on a keyboard | $n \times K$     |
| P        | Point with mouse to a target on the display   | 1.1              |
| B        | Press or release mouse button                 | 0.1              |
| BB       | Click mouse button                            | 0.2              |
| H        | Moving the hand between keyboard and mouse    | 0.4              |

(a) Keystroke-Level Model

| Task ID | Set of Task for QF            | Sequence of Actions  | Average Time (s)       |
|---------|-------------------------------|----------------------|------------------------|
| T1      | Add non-join predicate        | <A1, A2, A3, A4, A5> | $A1+A2+A3+A4+A5 = 9.9$ |
| T2      | Add join predicate            | <A1, A1, A2>         | $2A1+A2 = 3.6$         |
| T3      | Combine predicate with AND/OR | <A9,A10,A5>          | $A9+A10+A5 = 3.8$      |
| T4      | Add a RETURN clause element   | <A1,A2>              | $A1+A2 = 2.4$          |

(c) Average execution times for query formulation tasks

| Undo ID | Task                                                   | Sequence of Actions                    | Average Time (s) |
|---------|--------------------------------------------------------|----------------------------------------|------------------|
| U1      | Modify the LHS of a non-join predicate                 | <A1, A2, A5>                           | $A1+A2+A5 = 3.7$ |
| U2      | Modify the RHS of a non-join predicate                 | <A4, A5>                               | $A4+A5 = 6$      |
| U3      | Modify the comparison operator of a non-join predicate | <A3, A5>                               | $A3+A5 = 2.8$    |
| U4      | Modify the LHS or RHS of a join predicate              | <A1, A2, A5>                           | $A1+A2+A5 = 3.7$ |
| U5      | Change a AND to a OR (or vice versa)                   | <A10>                                  | $A10 = 1.3$      |
| U6      | Deleting a predicate/RETURN clause                     | <A5> (Click delete button in UNDO box) | $A5=1.3$         |

(d) Average execution times for UNDO tasks

| Action ID | Atomic Actions                                            | Sequence of physical operator                                                                                                                                                                         | Average Time (s)                   |
|-----------|-----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------|
| A1        | Select predicate node                                     | (a) Move the mouse on the node (P)<br>(b) Press mouse button (B)                                                                                                                                      | $P+B=1.2$                          |
| A2        | Drag and drop predicate node                              | (a) Move the mouse to Query Editor (P)<br>(b) Release mouse button (B)                                                                                                                                | $P+B=1.2$                          |
| A3        | Selection of comparison condition in condition dialog box | (a) Move the mouse to V button (P)<br>(b) Click mouse button (BB)<br>(c) Click mouse button on selected condition (BB)                                                                                | $P+2BB=1.5$                        |
| A4        | Type comparison value (avg 10 characters)                 | (a) Move the mouse to text box (P)<br>(b) Moving the hand between keyboard and mouse (H)<br>(c) Type characters (T(10))<br>(d) Moving the hand between keyboard and mouse (H) [for subsequent action] | $P+2H+T(10)=1.1 + 0.8 + 2.8 = 4.7$ |
| A5        | Click on a button in the combo box                        | (a) Move the mouse on the button (P)<br>(b) Click mouse button (BB)                                                                                                                                   | $P+BB=1.3$                         |
| A6        | Select action to UNDO                                     | (a) Move the mouse to UNDO icon (P)<br>(b) Click mouse button (BB)                                                                                                                                    | $P+BB=1.3$                         |
| A7        | Click on UNDO                                             | (a) Move the mouse to the action (P)<br>(b) Click mouse button (BB)                                                                                                                                   | $P+BB=1.3$                         |
| A8        | Click on RUN                                              | (a) Move the mouse to RUN icon (P)<br>(b) Click mouse button (BB)                                                                                                                                     | $P+BB=1.3$                         |
| A9        | Drag predicate in Query Editor (for AND/OR clause)        | (a) Drag mouse to other predicate (P)<br>(b) Release mouse (B)                                                                                                                                        | $P+B=1.2$                          |
| A10       | Select AND/OR operator                                    | (a) Move mouse on the AND or OR icon (P)<br>(b) Click mouse button (BB)                                                                                                                               | $P+BB=1.3$                         |

(b) Average execution times for atomic actions

**Fig. 2.** Query formulation times using Keystroke-Level model

query. Addition of *mental operators* while formulating a query will only increase the QFT and consequently increase the performance gain achieved due to prefetching. In other words, in this paper we investigate the benefits of prefetching for “worst case” QFT (without mental operators).

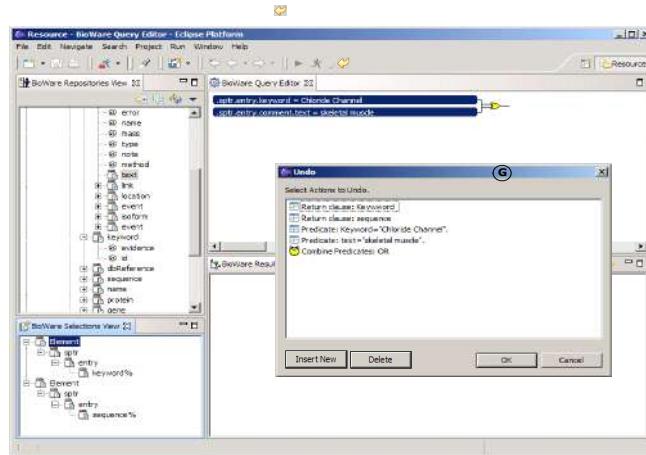
We first compute QFT in the absence of any query formulation error committed by the user. We call such QFT as *error-oblivious query formulation time* (EO-QFT). Note that our model for calculating the QFT can as well be used for other types of visual XML query formulation systems (such as XQBE [II]). This is because similar actions would be required to formulate a query.

### 3.1 Error-Oblivious QFT (EO-QFT)

Based on the timings (Figures 2(b) and 2(c)) discussed above the EO-QFT (denoted as  $T_f$ ) for a query can be calculated as follows:

$$T_f = 9.9(x_{nj} - 1) + 3.6x_j + 3.8b + 1.3 \quad (1)$$

where  $x_{nj}$  is the number of non-join predicates,  $x_j$  is the number of join predicates,  $b$  is the number of boolean operators in the query, and 1.3s is the time taken to click on the “Run” icon (Action A8 in Figure 2(b)). Observe that  $(x_{nj} - 1)$  is used as prefetching can start only when the first query formulation step is complete in the *Query Editor*. That is, QFT does not include the time taken to add the RETURN clause. This is because if prefetching were to start as soon as the RETURN clause were added, it is possible to retrieve very large results many of which may not be relevant eventually as WHERE clause predicates are yet to be added in the *Query Editor*. Fortunately, as we shall in Section 5, we achieve significant performance improvement even though we postpone the prefetching till addition of a WHERE clause predicate in the *Query Editor*.



**Fig. 3.** Undo operation

### 3.2 Error-Conscious QFT (EC\_QFT)

The above approach used to calculate error-oblivious QFT does not take into account errors committed by the user. These errors are referred to as *query formulation errors* (QFE). Note that QFEs may impact our prefetching approach. Hence, it is necessary to quantify the effect of QFEs by extending EO\_QFT with the time lost due to QFEs. We first discuss how the GUI enables the user to correct queries by undoing certain actions. Then, we compute the *error-conscious* QFT (EC\_QFT) that incorporates QFE.

Figure 3 shows the interface presented to the user. When the user discovers a mistake he/she clicks on the UNDO icon (labeled F in Figure 1(a)). The user is then presented with the list of actions he/she has performed (labeled G in Figure 3). For example, in Figure 3 the list shows that the user has added two predicates and combined them using a conjunction. The user then selects the action(s) to be corrected. Suppose the user wanted the second predicate to be `.sptr.entry.comment.text="cardiac muscle"` instead of `.sptr.entry.comment.text="skeletal muscle"` in Figure 3. Consequently, the user has to modify the predicate by replacing "skeletal muscle" with "cardiac muscle". In general, a user will execute the following steps to rectify a mistake.

**Step 1 (Click on the UNDO icon):** This takes 1.3s (A7 in Figure 2(b)).

**Step 2 (Select the action(s) to modify):** The user may select an action to update or delete by clicking on it or he/she may click the “Insert” button to insert new predicate(s) in the WHERE and RETURN clauses. Each action selection for update or delete will take at most 1.3s (A6 in Figure 2(b)). As there can be  $k$  number of actions to be modified, the total time will be  $1.3k$  seconds. The time taken to click “Insert” button is 1.3s (A5 in Figure 2(b)). If there are  $i$  such clicks then the total time is  $1.3i$ . The time taken to insert new non-join/join predicate(s) is  $(9.9i_{nj} + 3.6i_j)$  (Equation 1). Note that addition of AND/OR operators will be included by Step 3. The time taken to insert  $r$  RETURN

clause elements is also  $2.4r$  ( $T_4$  in Figure 2(c)). If “Insert” button is pressed then Step 3 is ignored by the user.

**Step 3:** In this step, some of the actions in Figure 2(d) need to be taken if the user selects action(s) for update or delete.

**Step 4 (Click on “OK” to accept the changes):** This will take  $1.3s$  ( $A_5$  in Figure 2(b)) and will have to be done for each modification. As a result, the total time taken for this operation is  $1.3 \times \mathfrak{R}$  where  $\mathfrak{R} = (i_{nj} + i_j + r + p_\ell + p_r + p_c + p_j + p_d + p_b)$  and  $p_\ell, p_r, p_c, p_j, p_d, p_b$  are numbers of times corrections  $U_1, U_2, U_3, U_4, U_5$ , and  $U_6$  in Figure 2(d) are made respectively.

**Step 5 (Click on “OK” button in Figure 3):** This takes  $1.3s$  ( $A_5$  in Figure 2(b)).

Therefore, *each* time the UNDO icon is clicked and a set of mistakes is corrected, the *additional* time taken for formulating a query will be  $(2.6 + 1.3k + 1.3i + T_u)$  where  $0 < k \leq \mathfrak{R}, i \geq 0$  and

$$\begin{aligned} T_u &= 9.9i_{nj} + 3.6i_j + 2.4r + 3.7p_\ell + 6p_r + 2.8p_c + 3.7p_j + 1.3p_d + 1.3p_b + 1.3\mathfrak{R} \\ &= 11.2i_{nj} + 4.9i_j + 3.7r + 5p_\ell + 7.3p_r + 4.1p_c + 5p_j + 2.6p_d + 2.6p_b \end{aligned} \quad (2)$$

The query formulation time  $T_f$  can now be extended to incorporate QFEs. If the user clicks on UNDO  $n$  times and corrects a set of mistakes each time then *error-conscious query formulation time* (denoted as  $T_{fe}$ ) is given by the following equation.

$$T_{fe} = 9.9(m_{nj} - 1) + 3.6m_j + 3.8m_b + \sum_{s=1}^n (2.6 + 1.3i_s + 1.3k_s + T_{us}) + 1.3(3)$$

where  $k_s, i_s$  and  $T_{us}$  are the number of actions to be modified, the number of times “Insert” button is selected, and the total time taken to correct the mistakes respectively, for the  $s^{th}$  instance of the UNDO operation. The variables  $m_{nj}$ ,  $m_j$ , and  $m_b$  are the number of non-join predicates, number of join predicates, and the number of boolean operators *correctly* added during query formulation respectively. Note that  $m_{nj}$ ,  $m_j$ , and  $m_b$  do not include those predicates and boolean operators that contain mistakes or inserted/deleted during UNDO operation.

## 4 GUI-Based Prefetching

We now describe our approach to improving query performance by utilizing the latency offered by GUI-based query formulation. Given an XML document and a path expression  $P$  the *Path Count* (denoted as  $C(P)$ ) is defined as the number of leaf nodes that satisfy  $P$ . The  $C(P)$  value for a non-root-to-leaf path  $P$  is  $\sum_{j=1}^k C(P_j)$  where  $P_1, P_2, \dots, P_k$  are the root-to-leaf paths that satisfy  $P$ . Note that, as  $C(P)$  increases so does the I/O cost of a query that contains  $P$  as one of its path expressions. The *Total Path Count* for an XML document is defined as  $T = \sum_{j=1}^N C(P_j)$  where  $N$  is the number of distinct root-to-leaf paths in the XML document. Next, we define the notion of *value selectivity*. Given an XML document and a root-to-leaf path  $P$ , *value selectivity*  $V(P)$  is defined as the number of nodes in the XML document with path  $P$  that have unique text values.

```

Input: Actions from the query interface.
Output: Intermediate materializations.
1: State $S = \text{getGUIState}()$
 /*prefetch till user executes query*/
2: while $S \neq \text{"Execute Query"}$ do
 /*Call materialization selection algorithm*/
3: selectMaterialization()
 /*Call materialization replacement algorithm*/
4: replaceMaterialization()
5: $S_i = \text{getGUIState}()$
6: while $S_i == S$ do /*wait till GUI state changes*/
7: $S_i = \text{getGUIState}()$
8: end while
9: end while

```

```

Input: Expressions $K = \{K_1, K_2, \dots, K_n\}$ in descending order of $\text{cost}(K_i)$.
 Materialization limit L_w .
Output: Coefficient of each K_i in the final materialization.
1: $\text{start} = 0$, $\text{end} = 2^n - 1$, middle
2: while $\text{start} < \text{end}$ do
3: $\text{middle} = (\text{start} + \text{end}) / 2$
4: /*GetSelection(order, n) generates
 the coefficients for orderth combination out of $2^n - 1$.*/
5: $S = \text{GetSelection}(\text{middle}, n)$
6: $I_k = \sum_{i=1}^n s_i \times \text{cost}(K_i)$
7: if $I_k > L_w$ then
8: $\text{end} = \text{middle} - 1$
9: else if then
10: $\text{start} = \text{middle} + 1$
11: end if
12: end while
13: return $\text{GetSelection}(\text{middle}, n)$

```

(a) Prefetching algorithm.

(b) Algorithm selectMaterialization.

**Fig. 4.** Algorithms for prefetching

Based on the above definitions, the cost of evaluating a `QueryExpr`  $\kappa$ , denoted as  $\text{cost}(\kappa)$ , can be calculated as follows. (1) If  $\kappa ::= \text{PathExpr}$  (`ValueComp`) `Literal` then the usual procedure to estimate the I/O cost is followed. When `Value Comp` is " $=$ " or " $>$ "  $\text{cost}(\kappa) = C(P)/V(P)$  or  $\text{cost}(\kappa) = C(P)/3$  respectively [5], where  $P$  is the parameter of type `PathExpr`. This can be extended to other types of `ValueComp`. (2) If  $\kappa ::= \text{PathExpr}$  (`ValueComp`) `PathExpr` and the two `PathExpr` types are denoted as  $P_1$  and  $P_2$  then  $\text{cost}(\kappa) = \frac{C(P_1)}{V(P_1)} \times \frac{C(P_2)}{V(P_2)}$ . (3) If  $\kappa ::= \text{ComparisonExpr}$  ( $\wedge$ ) `ComparisonExpr` and the two `ComaprisonExpr` types are denoted as  $\kappa_1$  and  $\kappa_2$  then the probability that  $\kappa_i$  ( $i = 1, 2$ ) is satisfied is  $\frac{\text{cost}(\kappa_i)}{T}$ . Therefore,  $\text{cost}(\kappa) = \frac{\text{cost}(\kappa_1) \times \text{cost}(\kappa_2)}{T}$ . (4) If  $\kappa ::= \text{ComparisonExpr}$  ( $\vee$ ) `ComparisonExpr` and the two `ComaprisonExpr` types are denoted as  $\kappa_1$  and  $\kappa_2$  then  $\text{cost}(\kappa) = \text{cost}(\kappa_1) + \text{cost}(\kappa_2)$ . Note that the last two formulae can be extended for any number of conjunctions and disjunctions.

#### 4.1 Prefetching Algorithm

The basic idea we employ for prefetching is that we prefetch constituent path expressions, store the intermediary results, reuse them when connective is added or "Run" is pressed. To realize this, the prefetching algorithm needs to perform prefetching operations at *certain* steps. In order to perform these operations, prefetching friendly GUI actions need to be identified first. Recall from Section 2, when a user formulates a query, constructs of types `QueryExpr` and `ComparisonExpr` are created. These types are parts of the final query and, therefore, are candidates for temporary materializations. Therefore, GUI actions that result in the addition of these types are also indicators for prefetching. These actions are: (1) the addition of an `ExprBox` and (2) combining two or more `ExprBox` types to create another `ExprBox` type that corresponds to a `QueryExpr` type.

Next, given a GUI state, *the optimal prefetching operations need to be determined*. Finally, since each prefetching operation is useful for the next, *existing materializations need to be replaced with new materializations preferably using the previous materializations*. Figure 4(a) shows the overall prefetching algorithm. The process continues till the user clicks on "Run" to execute the query (line 2). The process waits for changes in

|                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Input:</b> GUI state, last GUI operation op and current set of materializations $M$ .<br><b>Output:</b> Updated $M$ . | <pre> 21:   <math>M = M \cup m</math> 22:   <b>end if</b> 23:   <b>if</b> <math>\neg m_1</math> or <math>\neg m_2</math> <b>then</b> /* <math>m_1</math> or <math>m_2</math> or both are not materialized yet */ 24:     <math>M = \text{selectMaterialization}()</math> 25:     <b>if</b> <math>(m_1 \wedge m_2) \in M</math> <b>then</b> 26:       /* Use <math>m_1</math> or <math>m_2</math> to generate the new SQL query */ 27:       <math>SQl\ s = \text{SQL query using only } m_1 \text{ and } m_2.</math> 28:       <math>M = M - (m_1 \text{ and } m_2)</math> 29:       materialize s. 30:       <math>M = M \cup s</math> 31:     <b>end if</b> 32:   <b>end if</b> 33: <b>end if</b> 34: <b>if</b> op is UNDO <b>then</b> 35:   Cancel ongoing materialization. 36:   <math>M_c = \text{completed materializations dependant on step being corrected.}</math> 37:   <b>for all</b> <math>m_0 \in M_c</math> <b>do</b> 38:     Delete <math>m_0</math>. 39:   <b>end for</b> 40: <b>end if</b> 41: /*materialize the new ComparisonExpr or QueryExpr types in <math>M^*</math>/ 41: <b>materializeNew()</b>       </pre> |
|--------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Fig. 5.** Algorithm replaceMaterialization

the user interface (lines 5 to 8) before selecting new materializations (line 3). Once new materializations are selected, existing ones are replaced (line 4).

**Materialization Selection:** At any given step during query formulation there can be more than one materialization option. Therefore, an algorithm that selects the “best” materialization is required. We begin by presenting two heuristics that are used in our algorithm.

**Heuristic 1:** We consider only disjunctions of `ComparisonExpr` and `QueryExpr` as candidates for temporary materializations. We elaborate on the rational behind this heuristic now. While formulating queries the GUI contains  $n$  `ComparisonExpr` and `QueryExpr` types (denoted as  $\kappa_i$  where  $i = 1 \dots n$ ). Then, the possible materializations are  $(\kappa_1 \vee \kappa_2 \vee \kappa_3 \vee \dots \vee \kappa_n)$ ,  $(\kappa_1 \wedge \kappa_2 \vee \kappa_3 \vee \dots \vee \kappa_n)$ ,  $(\kappa_1 \wedge \kappa_2 \wedge \kappa_3 \vee \dots \vee \kappa_n)$  and so on. The number of possible combinations is  $2^{n-1}$ . Obviously, evaluating all possible materializations, though guaranteed to generate a useful materialization, is not feasible. Therefore, only disjunctions are generated. This is because given  $\kappa_1, \dots, \kappa_n$ ,  $(\kappa_1 \wedge \kappa_2 \wedge \dots \wedge \kappa_n)$  can be evaluated from the materialization of  $(\kappa_1 \vee \kappa_2 \vee \dots \vee \kappa_n)$ .

**Heuristic 2:** Given a materialization space limit  $L_M$ , we include the *maximum possible* number of expressions  $\kappa_i$  in the materialization. This is because the greater the number of expressions included in the current materialization the greater the usefulness of the intermediate result towards evaluating the final result.

Based on the above heuristics we define the notions of *materialization selection* and the *optimality* of a materialization selection. Given  $\kappa_1, \kappa_2 \dots \kappa_n$ , a *materialization selection* is defined as  $S = \{\mu_1, \mu_2, \dots, \mu_n\}$  where  $\mu_i \in \{0, 1\}$  and the cost associated with the selection (which is the same as the result size) is calculated as  $l_S = \sum_{i=1}^n \text{cost}(\kappa_i) \times \mu_i$ . Essentially, an expression  $\kappa_i$  is included in the materialization if  $\mu_i = 1$ . The cost  $l_S$  is a summation as only disjunctions are considered based on Heuristic 1.

The *optimality* of a materialization selection, denoted as  $\Theta(S)$ , is defined as follows. Given two materialization selections  $S_a = \langle \mu_{a_1}, \mu_{a_2}, \dots, \mu_{a_n} \rangle$  and  $S_b = \langle \mu_{b_1}, \mu_{b_2}, \dots, \mu_{b_n} \rangle$ ,  $\Theta(S_a) > \Theta(S_b)$  if and only if  $(\sum_{i=1}^n \mu_{a_i} > \sum_{i=1}^n \mu_{b_i}) \vee (\sum_{i=1}^n \mu_{a_i} = \sum_{i=1}^n \mu_{b_i} \wedge l_{S_a} > l_{S_b})$ . This optimality condition satisfies Heuristic 2. We elaborate on the usefulness of this with an example. Consider a GUI state with three expressions  $\kappa_1, \kappa_2$  and  $\kappa_3$  such that  $cost(\kappa_1) > cost(\kappa_2) > cost(\kappa_3)$ . The most desirable materialization selection would be  $S = \{1, 1, 1\}$  as it will include all the expressions. However, if  $l_S > L_M$  then selections with only two expressions will have to be considered. Then, the optimal materialization would be  $S = \{1, 1, 0\}$  as it includes the expressions that will yield the largest result. This can be extended to generate the sequence  $\Theta(\{1, 1, 1\}) > \Theta(\{1, 1, 0\}) > \Theta(\{1, 0, 1\}) > \dots > \Theta(\{0, 0, 1\}) > \Theta(\{0, 0, 0\})$ . Note that this sequence can be generated for any number of expressions  $n$ .

The algorithm is shown in Figure 4(b). The input to the algorithm is the list of ComparisonExpr and QueryExpr types,  $\kappa_i$ , currently present in the GUI. They are listed in decreasing order of  $cost(\kappa_i)$  as discussed above. Essentially, the algorithm performs a binary search over this sequence to determine the best materialization given the limit  $L_M$ . Notice that the sequence need not be pre-generated. The *GetSelection* method returns a selection  $S$  given its order in the sequence and the number of expressions  $n$ . For example, in the case where  $n = 3$ , *GetSelection*(3, 3) would return  $\{1, 0, 1\}$  - the third selection for three rules. Similarly, *GetSelection*(1, 3) would return  $\{1, 1, 1\}$ . It can be shown that the overall time complexity of the algorithm is  $O(n^3)$ . Once the list of expressions is selected by the algorithm a separate materialization, denoted as  $\mathcal{M}_{\kappa_i}$ , is maintained for each  $\kappa_i$ . Note that a disjunction of the selected  $\kappa_i$ s could be maintained instead. However, the cost for both is approximately the same and is equal to  $\sum cost(\kappa_i)$ .

**Materialization Replacement:** Once the optimal materialization to replace the current state is selected it needs to be generated preferably using the results from the previous materializations. The materialization replacement algorithm is presented in Figure 5. The worst case complexity of the replacement algorithm without executing the new materialization is  $O(n^3)$  - when *selectMaterialization()* is called. The overall time taken depends on the execution time the SQL query(s) corresponding to the new materialization.

## 5 Performance Study

The prototype system of GUI-driven prefetching technique was implemented using JDK1.5. The visual interface was built as a plug-in for the Eclipse platform ([www.eclipse.org](http://www.eclipse.org)). The RDBMS used was SQL Server 2000 running on a P4 1.4GHz machine with 256MB RAM. As mentioned in Section 1, our approach can be built on any XML-to-relational storage mechanism. In this paper, we have adopted our schema-oblivious XML storage system called SUCXENT++ [8].

The experiments were carried out with three data sets of size 300MB, 600MB and 1200MB respectively generated by combining the data sets shown in Figure 6(a). The 300MB data sets was generated using 150MB each of the SWISS-PROT and EMBL data sets. The 600MB data set was generated using 300MB each and the 1200MB data set

| Data       | URL                  | Size (MB) | Node Count | Leaf Count | Depth |
|------------|----------------------|-----------|------------|------------|-------|
| Swiss-Prot | http://us.expasy.org | 600       | 26,035,096 | 17,385,288 | 6     |
| EMBL       | http://ebi.ac.uk     | 600       | 15,265,784 | 13,460,524 | 6     |
| Enzyme     | http://ebi.ac.uk     | 3         | 86,413     | 74,892     | 7     |
| Total      |                      | 1203      | 41,387,293 | 30,920,704 |       |

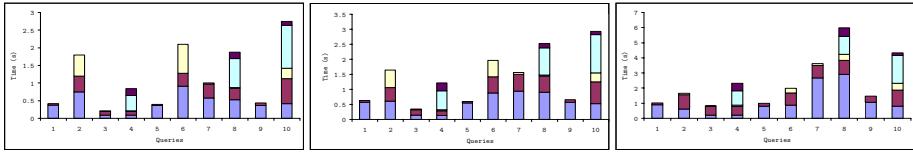
(a) Data Set

| #  | Query                                                                                                                                                                                  | Characteristic                                                                                    | T <sub>r</sub> (s) | Result Size |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|--------------------|-------------|
| Q1 | for \$b in /sptr/entry<br>where \$b/protein/name = 'Sesquiterpene<br>Cyclase'<br>return \$b/accession                                                                                  | - Database: Swiss-Prot<br>- single non-join predicate<br>- small result size                      | 1.3                | 3           |
| Q2 | for \$b in /sptr/entry/<br>where \$b/feature[@type = 'transmembrane<br>region'] and \$b/organism/name = 'human'<br>return \$b/accession                                                | - Database: Swiss-Prot<br>- two non-join predicates<br>- AND operator<br>- large result size      | 11.7               | 2838        |
| Q3 | for \$b in /sptr/entry/<br>where (\$b/comment/text = 'Chloride Channel'<br>or \$b/comment/text = 'skeletal muscle')<br>return \$b/accession,\$b/sequence                               | - Database: Swiss-Prot<br>- two non-join predicates<br>- OR operator<br>- small result size       | 14.8               | 145         |
| Q4 | for \$b in /sptr/entry/<br>where (\$b/keyword = 'Chloride Channel'<br>or \$b/comment/text = 'skeletal muscle')<br>and \$b/organism/name = 'human'<br>return \$b/accession,\$b/sequence | - Database: Swiss-Prot<br>- three non-join predicates<br>- AND/OR operator<br>- small result size | 24.9               | 43          |

| #   | Query                                                                                                                                                                                                                                                                                           | Characteristic                                                                                   | T <sub>r</sub> (s) | Result Size |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|--------------------|-------------|
| Q5  | for \$b in /embly/entry<br>where \$b/keyword = "%gene%"<br>return \$b/accession                                                                                                                                                                                                                 | - Database: EMBL<br>- single non-join predicate<br>- large result size                           | 1.3                | 3349        |
| Q6  | for \$b in /embly/entry<br>where \$b/source/organism='Homo Sapiens'<br>and \$b/keyword = "%gene%"<br>return \$b/accession                                                                                                                                                                       | - Database: EMBL<br>- two non-join predicates<br>- AND operator<br>- large result size           | 11.98              | 3278        |
| Q7  | for \$b in /embly/entry<br>where \$b/descr = "%gene%" or \$b/keyword<br>= "%gene%"<br>return \$b/accession                                                                                                                                                                                      | - Database: EMBL<br>- two non-join predicates<br>- OR operator<br>- large result size            | 11.98              | 3683        |
| Q8  | for \$b in /embly/entry<br>where (\$b/descr = "%gene%" or \$b/keyword<br>= "%gene%") and \$b/source/organism =<br>"Homo Sapiens"<br>return \$b/accession                                                                                                                                        | - Database: EMBL<br>- three non-join predicates<br>- AND/OR operator<br>- large result size      | 24.34              | 3596        |
| Q9  | for \$b in /embly/entry, \$c in /embly/entry<br>where \$b/organism/name = 'Sesquiterpene<br>Cyclase' and \$b/dbReference[@id=\$c/_id]<br>accession)<br>return \$b/accession, \$c/accession                                                                                                      | - Database: Swiss-Prot,<br>EMBL<br>- single join predicate<br>- AND operator                     | 8.7                | 3           |
| Q10 | for \$b in /sptr/entry, \$c in /<br>enzyme_pathway/entry, \$d in /embly/entry<br>where \$d/keyword = "%gene%" and \$b/<br>accession=\$c/swissprot_reference/<br>reference[\$d/references/@type='EMBL']<br>and \$b/dbReference[@id=\$d/_id]<br>accession)<br>return \$b/accession, \$c/accession | - Database: Swiss-Prot,<br>Enzyme and EMBL<br>- two join predicates<br>- three Boolean operators | 26.22              | 68          |

(b) Queries

Fig. 6. Data set and queries



(a) 300 MB

(b) 600 MB

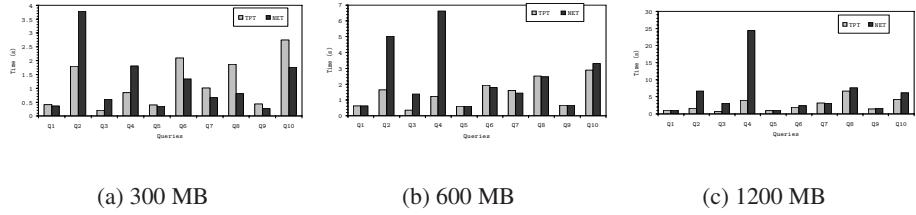
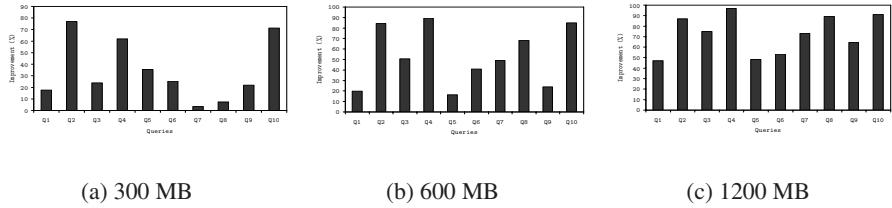
(c) 1200 MB

Fig. 7. Materialization replacement cost

was generated using the complete data sets. The 3MB ENZYME data set was used in all experiments. It is not reflected in the respective sizes due to its much smaller size. Ten queries were used to test the system. The list of queries together with their *EO\_QFT* values and query results size for 1200MB data is shown in Figure 6.

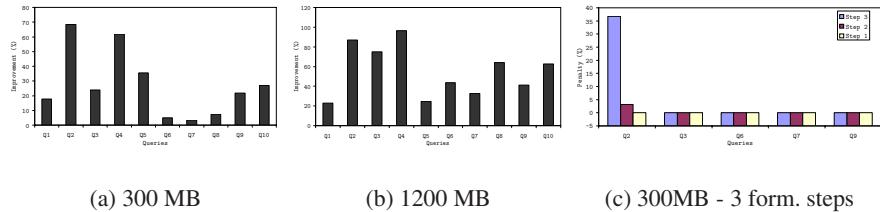
We now define few terms that are used in the subsequent discussion. The response time as perceived by the user when prefetching is not employed is called the *normal execution time (NET)* (denoted as  $T_n$ ). The *perceived response time (PRT)* is the query response time when prefetching is employed. In the absence of QFEs, we refer to the PRT as *error-oblivious* perceived response time (*EO\_PRT*). If QFEs are present then we refer to the PRT as *error-conscious* perceived response time (*EC\_PRT*). The total time taken for all prefetching operations is called *total prefetching time (TPT)*. Next we define the notion of *error realization distance*. Consider a query with  $n$  formulation steps where the user clicks on “Run” at  $n$ th step. Suppose that the error is committed at  $p$ th step and the UNDO operation is invoked at  $q$ th step where  $0 < p < q \leq n - 1$ . Then, the *error realization distance*, denoted as  $\epsilon$ , is defined as  $\epsilon = q - p$ .

**Materialization Replacement Cost:** Figure 7 shows the results of materialization replacement cost. Here the running times of individual materialization operations are presented. Each section of the stacked columns represents the running time associated with

**Fig. 8.** NET vs. TPT**Fig. 9.** NET vs EO\_PRT

the corresponding materialization. For example, Q1 has two formulation steps and, therefore, two sections in the corresponding stacked column. There are two main observations. First, the increase in the running times as the data set size increases is less than linear. Therefore, the cost associated with materialization replacement is scalable. Second, the replacement cost for disjunctions is less than that for conjunctions. This is reflected in the results for queries involving disjunction (Q3, Q4, Q7 and Q8) as opposed to queries involving conjunction (Q2, Q6, Q9 and Q10). This is expected as the materialization selection algorithm selects materializations with disjunctions (Heuristic 1). As a result, evaluating conjunctions would involve an additional step.

**NET vs TPT:** This experiment is required to test the viability of prefetching. Figure 8 shows the results for this experiment. There are three main observations. The first is that the difference is not significant indicating that prefetching is a viable option. The second observation is that the conjunctive queries show a smaller difference than disjunctive ones. This is because conjunctive queries are evaluated from the corresponding disjunction based on materialization selection/replacement algorithms. This means that conjunctive queries will have a more significant prefetching overhead. This observation can be extended to queries that proceed from less selective partial queries to more selective final queries during formulation. The final observation is that for some of the queries (e.g., Q2, Q4, Q10), interestingly, the sum of the prefetching operations is less than the actual query execution time. This difference increases with data set size. This can be explained as follows. The search phase during query optimization typically treats the estimated cost model parameter values as though they were completely precise and accurate, rather than the coarse estimates that they actually are. Consequently, the relational query optimizer may fail to produce query plans that are more robust to estimation errors especially for complex queries. For Q2, Q4, and Q10, individual prefetching



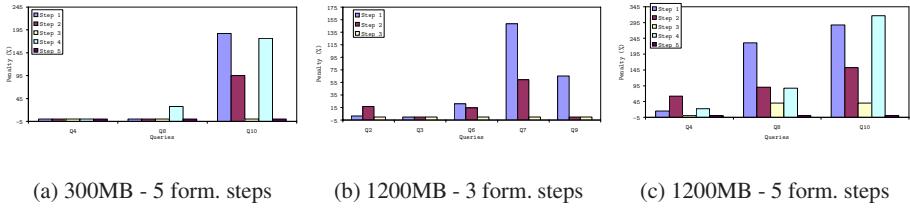
**Fig. 10.** EC\_PRT vs NET and EC\_PRT vs EO\_PRT(1)

queries are relatively simpler compared to a single normal query. Hence, we observe such response time.

**NET vs EO\_PRT:** The next experiment compares the *NET* with the *error-oblivious* perceived response time. This comparison is done as a percentage of improvement over normal execution. It is measured as  $improvement = (1 - \frac{EO\_PRT}{NET}) \times 100$ . Figure 9 show the results for the three data sets. There are two main observations. First, the improvement in performance is more for larger data sets. For the 300MB data set the improvement range is 7-76%. This range increases to 16-89% for the 600MB data set and 47-96% for the 1200MB data set. The second observation is that simple queries (Q1, Q5 and Q9) with one predicate and small result sets benefit the least. Queries with multiple predicates and large result sets benefit the most. This is indeed encouraging as query response time is more critical for large data set. Also queries with disjunctions benefit more than the queries with conjunctions. This is expected as the materialization selection algorithm selects disjunctions as the intermediate results. Q2 seems to go against this observation. As mentioned earlier, this is due to the wide gap in the optimality of the query plans generated in the two approaches.

**NET vs EC\_PRT:** In this experiment we evaluate the effect of QFE on perceived response time over normal execution time. This comparison is done as a percentage of improvement over normal execution. It is measured as  $improvement = (1 - \frac{EC\_PRT}{NET}) \times 100$ . In this experiment we present the *worst-case* value for *EC\_PRT* as discussed in [2]. The results are presented in Figures 10(a) and 10(b). We only take the smallest and the largest data sets (300MB and 1200MB) for this experiment. The main observation is that *EC\_PRT* is still significantly better than *NET* for most queries. Also observe that similar to *EO\_PRT*, there is larger improvement for larger data size. Hence, QFEs do not significantly affect the performance improvement achieved by GUI-driven prefetching.

**EC\_PRT vs EO\_PRT:** This comparison is done to measure the penalty on PRT due to QFE. It is measured as  $penalty = \frac{EC\_PRT - EO\_PRT}{EO\_PRT} \times 100$ . Again, the worst case value of *EC\_PRT* is used for comparison. Particularly, we measure *EC\_PRT* for  $q = n - 1$  (UNDO operation invoked just before clicking “Run”) and vary error realization distance. Figures 10(c) and 11 show the results for the 300MB and 1200MB data sets. Figure 10(c) shows the results for queries that have three formulation steps (two predicates and a conjunction/disjunction) other than clicking on “Run” and Figure 11(a)

**Fig. 11.** EC\_PRT vs EO\_PRT (2)

shows the results for queries with five formulation steps. The three values shown for each query in Figure 10(c) measure the penalty when the error was committed at the first step, the second step and the third step respectively (variation of  $\epsilon$ ). The *penalty* axis starts at  $-5$  to allow the display of cases where  $penalty = 0$ .

The results shown highlight two main points. First, QFE generally has a greater effect with the increase in error realization distance. This is expected as an early mistake will lead to more materializations being recalculated. However, there are some exceptions. The query Q2 for the 1200MB data set shows an increase as the evaluation of the second predicate is more expensive than the first. Similar phenomenon is observed for query Q4. Second, the impact of QFE increases with data set size. The 1200MB data set shows a maximum increase of 316%. The 300MB data set shows a maximum increase of 187%. The impact of QFE is felt on only four queries for the 300MB data set whereas all queries are effected for the 1200MB data set. This can be attributed to the higher cost of reevaluating materializations for the larger data set.

## 6 Related Work

**GUI-latency driven optimization:** Closest to our work is the effort by Polyzotis et al. [7] in *speculative* query processing. The method described is for relational data and incorporates speculation where the final query (or sub-queries that will be present in the final query) is predicted based on the user's usage profile. Machine learning techniques are applied on past user actions and a user-behavior model is formulated. In comparison, our approach employs deterministic prefetching without speculating on the final form of the query. This could result in a less than maximum gain in certain cases but there are no penalties. Speculation can lead to execution time penalties when the prediction is incorrect. In our case, this problem does not arise. Furthermore, we do not need to keep track of user's usage profile, but still can achieve comparable query performance improvement.

**Prefetching and Caching:** To the best of our knowledge, we have not found any published work related to prefetching techniques for XML data. Closest to the prefetching approach is caching, which although investigated extensively in relational database systems, is a relatively new area of research for XML data. However, XML caching techniques mentioned in [9] operate on the final query and do not take into account the individual steps in query formulation. In our approach, partial queries are materialized at each formulation step by utilizing the latency offered by GUI-driven query

formulation. This presents a significant advantage over caching as every query benefits from prefetching unlike caching - where only those queries whose results have been cached improve in performance.

## 7 Conclusions and Future Work

The main contribution of this paper is to show that the latency offered by visual query formulation can be utilized to prefetch partial results so that the final query can be answered in a shorter time. We show that prefetching is viable as the combined time taken by all the prefetching operations is not significantly more than normal query execution time. In fact, for some queries the total time taken by all prefetching operations is less than the normal execution time due to a better query plan generated by the relational query optimizer. Our experiments also show that prefetching improves the perceived query response time by 7-96% with a greater improvement for larger data sets. In addition, query formulation errors have no significant influence on the perceived response time compared to the normal execution time. GUI-driven prefetching is potentially of value in XML query processing context where one would like to use a user-friendly GUI to formulate queries. Future directions of research include extension of our prefetching technique to more advanced XQueries, more sophisticated I/O cost estimation technique, and explore benefits of prefetching in a multiuser environment.

## References

1. E. AUGURUSA, D. BRAGA, A. CAMPI, S. CERI. Design and Implementation of a Graphical Interface to XQuery. In *ACM SAC*, 2003.
2. S. S. BHOWMICK AND S. PRAKASH. Efficient XML Query Processing in RDBMS Using GUI-driven Prefetching in a Single-User Environment. *Technical Report*, CAIS-03-2005, School of Computer Engg, NTU, 2005 (Available at <http://www.ntu.edu.sg/home/assourav/papers/cais-03-2005-TR.pdf>).
3. S. S. BHOWMICK AND S. PRAKASH. Every Click You Make, I Will be Fetching It: Efficient XML Query Processing in RDBMS Using GUI-driven Prefetching. In *ICDE*, 2006 (Poster paper).
4. S. K. CARD, T. P. MORAN, AND A. NEWELL. The Keystroke-level Model for User Performance Time with Interactive Systems. *Commun. ACM*, 23(7):396–410, 1980.
5. G. GRAEFE (ED.). Special Issue on Query Processing in Commercial Database Management Systems. *IEEE Data Engineering*, 16:4, 1993.
6. L. LUO AND B. E. JOHN. Predicting Task Execution Time on Handheld Devices Using the Keystroke-Level Model. In *ACM CHI*, 2005.
7. N. POLYZOTIS AND Y. IOANNIDIS. Speculative Query Processing. In *CIDR*, 2003.
8. S. PRAKASH, S. S. BHOWMICK, S. K. MADRIA. Efficient Recursive XML Query Processing Using Relational Databases. In *DKE*, 58(3), 2006.
9. L.-H. YANG, M.-LI. LEE, AND W. HSU. Efficient Mining of XML Query Patterns for Caching. In *VLDB*, 2003.

# Efficient Holistic Twig Joins in Leaf-to-Root Combining with Root-to-Leaf Way

Guoliang Li, Jianhua Feng, Yong Zhang, and Lizhu Zhou

Department of Computer Science and Technology, Tsinghua University,  
Beijing 100084, China  
{liguoliang, fengjh, dcszlz}@tsinghua.edu.cn,  
Zhangy@tsinghua.org.cn

**Abstract.** Finding all the occurrences of a twig pattern on multiple elements in an XML document is a core operation for efficient evaluation of XML queries. Holistic twig join algorithms, TwigStack and TSGeneric, have been recognized as optimal solutions when the twig pattern only involves A-D(ancestor-descendant) relationships, while iTwigJoin can be optimal for partial twig patterns that contain A-D only or P-C (parent-child) only relationships. However, existing algorithms involve unnecessary computations and CPU cost of them can be further improved, and we in this paper mainly address this problem. We first propose three effective optimization rules to avoid those unnecessary computations, and then present two algorithms incorporated with these optimization rules to effectively answer twig patterns in leaf-to-root combining with root-to-leaf way. Experimental results on various datasets indicate that our algorithms perform significantly better than existing proposals.

## 1 Introduction

XML is emerging as a de facto standard for information exchange over the Internet. Although XML documents could have rather complex internal structures, they can be modeled as rooted, ordered and labeled trees. Queries in XML query languages (e.g., XPath [BBC<sup>+</sup>02], XQuery [BCF<sup>+</sup>02]) typically specify patterns of selection predicates on multiple elements which have some specified structural relationships. For example, to retrieve all *paragraphs* satisfying the XPath: `//section//title//keyword]//paragraph[//figure]`. Such a query can be represented as a node-labeled twig pattern (or a small tree) with elements and string values as node labels [BKS02]. Finding all occurrences of a twig pattern is a core operation in XML query processing [FK99, STZ<sup>+</sup>99, TVB<sup>+</sup>02].

A typical approach is to first decompose the pattern into a set of binary structural relationships (P-C or A-D) between pairs of nodes, then match each of the binary structural relationships against the XML database, and finally stitch together the results from those basic matches [ZND<sup>+</sup>01, LM01, AJK<sup>+</sup>02, CVZ<sup>+</sup>02, JLW03, MHH06]. The main disadvantage of such a decomposition based approach is that intermediate result sizes can become very large, even if the input and the final result sizes are much more manageable.

To address this problem, many holistic twig join algorithms are proposed, such as TwigStack [BKS02], TSGeneric [JWL<sup>+</sup>03], TJFast [LLC<sup>+</sup>05], iTwigJoin [CLL05]. They answer the twig query holistically and avoid huge intermediate results. However they have to recursively call a subroutine *getNext* many times, which is a core function for the holistic algorithms. *getNext* always returns a node  $q$  and ensures that: (i) the current element in stream  $q$  has a descendant element in each stream  $q_i$ , for  $q_i \in \text{children}(q)$ , and (ii) each current element in stream  $q_i$  recursively satisfies the first property. If node  $q$  satisfies the two properties,  $q$  is called to have a solution extension, which will be introduced in detail in section 4. However, existing algorithms will call *getNext* many times, but most of them are unnecessary and could be avoided. Hence, they involve many unnecessary computations, and the potential benefit of CPU cost is not fully explored among those existing proposals. For example, if querying Q1 on the XML document in Fig.1(a), existing algorithms will call *getNext(s)*, *getNext(p)*, *getNext(f)*, *getNext(t)*, *getNext(k)* many times respectively, but only a few times are useful and pivotal, and other times can be pruned. Table 1 lists the main flow about how algorithm TwigStack works on the example in Fig.1. The flow is similar to those of other algorithms and merging the partial solutions is omitted.

However, there are three circumstances that some computations could be avoided:

- 1) ***Self-nested suboptimal.*** Given two nodes  $u_1, u_2$  with the same label, if  $u_1$  is an ancestor of  $u_2$ , we call they are self-nested. For example, in Fig.1, as  $s1$  is an ancestor of  $s2$ , which in turn is an ancestor of  $s3$ , so  $s1, s2$  and  $s3$  are self-nested. Since  $s1$  has a solution extension through calling *getNext* in steps 1-5, and when checking whether  $s2$  has a solution extension, we only need to check whether  $s2$  is a common ancestor of  $p1$  and  $t1$ , which are current elements of its child nodes  $p$  and  $t$ . However, it is unnecessary to recursively check whether  $p, t$  and their corresponding descendants, i.e.  $f, k$ , have solution extensions. The reason is that the streams of  $p, t$  and their descendants are not changed, and even if checking them again, *getNext* will return the same results as steps 1-5. Accordingly, existing algorithms involve self-nested suboptimal, and some unnecessary computations (e.g. steps 6-15) can be pruned.
- 2) ***Order suboptimal.*** If a node has more than one child, selecting which child to first check whether having a solution extension is important to twig joins, however existing algorithms do not consider this problem. For example, in Table 1, if node  $t$  (but not node  $p$ ) is first selected to check, it will return node  $k$  directly without involving to check  $p$  and  $f$ , subsequently steps 22,23,27,28 are unnecessary and can be pruned.
- 3) ***Stream null suboptimal.*** If stream  $q$  is empty, it is unnecessary to scan elements in the streams of  $q$ 's ancestors and descendants, because the elements in those streams will not contribute to final solutions. For example, in Fig.1, when the stream of node  $t$  is empty, it is unnecessary to check the elements of node  $s$  (the parent of  $t$ ) and  $k$  (a descendant of  $t$ ), thus steps 34, 35 in Table 1 are unnecessary. Even if there are some other elements in streams  $s$  and  $k$ , and *start* values of them are larger than *start* value of  $t1$ , they also can be skipped directly. Accordingly, if some streams are empty, elements in the streams of their ancestors and descendants can be skipped.

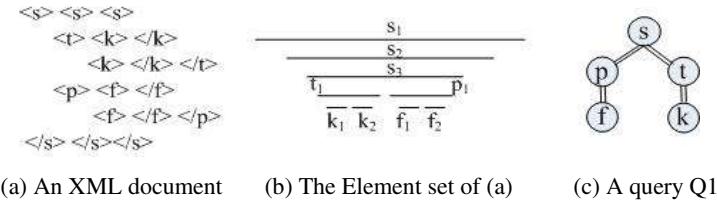


Fig. 1. An XML document and a query Q1

Table 1. The main flow about how TwigStack (or TSGeneric) works

| <i>getNext(s)</i> |              | <i>getNext(p)</i> |          | <i>getNext(f)</i> |          | <i>getNext(t)</i> |          | <i>getNext(k)</i> |          |
|-------------------|--------------|-------------------|----------|-------------------|----------|-------------------|----------|-------------------|----------|
| steps             | result       | steps             | result   | steps             | result   | steps             | result   | steps             | result   |
| 1                 | s(s1)        | 2                 | p        | 3                 | f        | 4                 | t        | 5                 | k        |
| <b>6</b>          | <b>s(s2)</b> | <b>7</b>          | <b>p</b> | <b>8</b>          | <b>f</b> | <b>9</b>          | <b>t</b> | <b>10</b>         | <b>k</b> |
| <b>11</b>         | <b>s(s3)</b> | <b>12</b>         | <b>p</b> | <b>13</b>         | <b>f</b> | <b>14</b>         | <b>t</b> | <b>15</b>         | <b>k</b> |
| 16                | t(t1)        | 17                | p        | 18                | f        | 19                | t        | 20                | k        |
| 21                | k(k1)        | <b>22</b>         | <b>p</b> | <b>23</b>         | <b>f</b> | 24                | k        | 25                | k        |
| 26                | k(k2)        | <b>27</b>         | <b>p</b> | <b>28</b>         | <b>f</b> | 29                | k        | 30                | k        |
| 31                | p(p1)        | 32                | p        | 33                | f        | <b>34</b>         | <b>t</b> | <b>35</b>         | <b>k</b> |
| 36                | f(f1)        | 37                | f        | 38                | f        |                   |          |                   |          |
| 39                | f(f2)        | 40                | f        | 41                | f        |                   |          |                   |          |

To discover a partial solution, existing algorithms do not consider the circumstances of self-nested, order and stream null suboptimal, and they have to check whether a sub-twig pattern has a solution extension many times and involve additional computations. In this paper, we will demonstrate how to avoid those unnecessary computations under these three circumstances through discovering the partial solutions in leaf-to-root combining with root-to-leaf way. We first propose three effective optimization rules to improve these three suboptimal and further explore the potential benefit of CPU cost, and subsequently present an algorithm, TJEssential, which avoids repeatedly checking whether a sub-twig pattern has a solution extension.

Our contributions can be summarized as follows:

- We propose three optimization rules to explore the potential benefit of CPU cost, which can avoid self-nested, order and stream null suboptimal of existing studies.
- We present an efficient holistic twig join algorithm, TJEssential, which discover all the solutions in leaf-to-root combining with root-to-leaf way. More importantly, we incorporate the three optimization rules into our algorithm to optimize twig joins.
- We implemented our proposed method and conducted an extensive performance study using both real and synthetic datasets of various characteristics. The results showed that our algorithm achieved high efficiency and outperformed existing proposals.

The rest of the paper proceeds as follows. Section 2 is dedicated to some related work. We introduce some notations in section 3. Then TJEssential algorithm is proposed in detail in section 4. Section 5 reports experimental results, and we conclude in section 6.

## 2 Related Work

In the context of semi-structured and XML databases, structural join was essential to XML query processing because XML queries usually imposed certain structural relationships. For binary structural join, Zhang et al. [ZND<sup>01</sup>] proposed a multi-predicate merge join (MPMGJN) algorithm based on  $(start, end, level)$  labeling of XML elements. Li et al [LM01] proposed  $\xi\xi/\xi A$ -Join. Stack-tree-Desc/Anc was proposed in [AJK<sup>02</sup>], and [CVZ<sup>02</sup>], [G02], [JLWO03] were index-based approaches. The later work by Wu et al [WPJ03] studied the problem of binary join order selection for complex queries on a cost model which took into consideration factors such as selectivity and intermediate results size.

Bruno et al [BKS02] proposed a holistic twig join algorithm, namely TwigStack, to avoid producing a large intermediate result. With a chain of linked stacks to compactly represent partial results of individual query root-to-leaf paths, TwigStack merged the sorted lists of participating element sets altogether, without creating large intermediate results. TwigStack has been proved to be optimal in terms of input and output sizes for twigs with only A-D edges. Further, Jiang et al [JWL<sup>03</sup>] studied the problem of holistic twig joins on all/partly indexed XML documents. Their proposed algorithms used indices to efficiently skip the elements that do not contribute to final answers, but their method can not reduce the size of intermediate results. Choi et al [CMW03] proved that optimality evaluation of twig patterns with arbitrarily mixed ancestor-descendant and parent-child edges was not feasible. Lu et al [LCL04] proposed the algorithm TwigStackList, which was better than any of previous work in term of the size of intermediate results for matching XML twig pattern with both P-C and A-D edges. Chen et al [CLL05] proposed an algorithm iTwigJoin, which was still based on region encoding, but worked with different data partition strategies (e.g. Tag+Level and Prefix Path Streaming), and Tag+Level Streaming can be optimal for both A-D and P-C only twig patterns whereas PPS streaming could be optimal for A-D only, P-C only and one branch node only twig patterns assuming there was no repetitive tag in the twig patterns. [LLC<sup>05</sup>] proposed a novel algorithm, TJFast on extended Dewey that only used leaf nodes' streams and saved I/O cost. More recently, Mathis et al. [MHH06] proposed a set of new *locking-aware* operators for twig pattern query evaluation to ensure data consistency, and Chen et al. [CLT<sup>06</sup>] presented Twig<sup>2</sup>Stack algorithm to avoid huge intermediate results. However, Twig<sup>2</sup>Stack reduced the intermediate results at the expense of a huge memory requirement, and it was restricted by the fan-out of the XML document.

## 3 Background

**XML data model and numbering scheme.** XML data is commonly modeled by a tree structure, where nodes represent elements, attributes and texts, and edges represent element-subelement, element-attribute and element-text pairs. Most existing XML query processing algorithms use a region code (*start*, *end*, *level*) to present the position of a tree node in the data tree. *start* and *end* are calculated by performing a pre-order traversal of the document tree; *level* is the level of a certain element in its data tree. The region encodings support efficient evaluation of structural relationships.

Formally, element  $u$  is an ancestor of element  $v$  if and only if  $u.start < v.start < u.end$ . For P-C relationship, it is also necessary to check whether  $u.level = v.level - 1$ .

**Twig pattern matching.** Queries in XML query languages make use of twig patterns to match relevant portions of data in an XML database. A twig pattern is a selection predicate on multiple elements in an XML document. Such query patterns can generally be represented as node-labeled trees. Twig pattern nodes may be elements, attributes and texts, and twig pattern edges are either P-C relationships (denoted by "/") or A-D relationships (denoted by "//"). If the number of children of a node is greater than one, we call this node a branching node. While if the node has only one child, it is a non-branching node. Matching a twig pattern against an XML database is to find all occurrences of the pattern in the database. Formally, given a twig pattern  $Q$  and an XML database  $D$ , a match of  $Q$  in  $D$  is identified by a mapping from nodes in  $Q$  to nodes in  $D$ , such that: (i) query node predicates are satisfied by the corresponding database elements; and (ii) the P-C and A-D relationships between query nodes are satisfied by the corresponding database elements. The answer (solution) to query  $Q$  with  $n$  nodes can be represented as a list of  $n$ -array tuples, where each tuple  $(q_1, q_2, \dots, q_n)$  consists of the database elements that identify a distinct match of  $Q$  in  $D$ .

**Notations.** Let  $q$  denote a twig pattern, as well as (interchangeably) the root node of the twig pattern. The self-explaining functions  $isRoot(q)$  and  $isLeaf(q)$  examine whether a query node  $q$  is a root or a leaf node. The function  $children(q)$  gets all child nodes and  $parent(q)$  returns the parent node of  $q$ . When there is no ambiguity, we may also refer to node  $q$  as the sub-query tree rooted at  $q$ . In the rest of this paper, "node" refers to a tree node in the twig pattern (e.g., node  $q$ ), while "element" refers to the elements in the dataset involved in a twig join. Let's assume there is a data stream associated with each node in the query tree. Every element in the data stream is already encoded in the following region format:  $(start, end, level)$ . Each data stream is already sorted on the  $start$  attribute. We also assume the join algorithms will make use of two types of data structures: cursors and stacks. Given a query tree  $Q$ , we associate a cursor ( $C_q$ ) and a stack ( $S_q$ ) to every node  $q \in Q$ . Each cursor  $C_q$  points to some element in the corresponding data stream of node  $q$ . Henceforth, " $C_q$ " or "element  $C_q$ " will refer to the element  $C_q$  points to, when there is no ambiguity. The cursor can move to the element (if any) next to element  $C_q$ . Such behavior can be invoked with  $C_q->advance()$ . We add  $nil$  to the end of each stream, and  $C_q$  points to  $nil$  (denoted as  $C_q=nil$ ) means all the elements of  $q$  have been processed. Similarly, we can access the attribute values of element  $C_q$  by  $C_q.start$ ,  $C_q.end$  and  $C_q.level$ . Initially, all the cursors point to the first element of the corresponding data stream, and all stacks are empty. We can access the top and bottom elements of  $S_q$  by  $S_q.top()$  and  $S_q.bottom()$ . During query execution, each stack  $S_q$  may cache some elements before the cursor  $C_q$  and these elements are strictly nested from bottom to top, i.e. each element is a descendant of the element below it. We also associate with each element  $e$  in  $S_q$  a pointer to the lowest ancestor in  $S_{parent(q)}$ . Thus, we can efficiently access all  $e$ 's ancestors in  $S_{parent(q)}$ . In fact, cached elements in stacks represent the partial solutions that could be further extended to full results as the algorithm goes on.

## 4 The TJEssential Algorithm

### 4.1 Preliminaries

We first introduce some concepts and then present three optimization rules in this section.

**Definition 1(Solution Extension).** *Node  $q$  has a solution extension if there is a solution for the sub-query rooted at  $q$  composed entirely of the current elements (cursors point to) of the query nodes in the sub-query [JWL<sup>+</sup>03].*

**Definition 2(Partial Solution).** *The tuple  $(e_1, e_2, \dots, e_n)$  consists of the database elements that identify a match of  $Q$  on document  $D$ , and the tuple  $(e_{i1}, e_{i2}, \dots, e_{ik})$  composed of elements of the query nodes on the path from the root to any leaf, is called a partial solution, where  $e_{i1}, e_{i2}, \dots, e_{ik}$  are elements in tuple  $(e_1, e_2, \dots, e_n)$ .*

**Definition 3(Quasi-Potential Element).** *Suppose a sub-query  $q$  of  $Q$  has a solution extension,  $q'$  is any non-leaf node in  $q$ , and  $\text{Startmin} = \min\{C_{qi}.\text{start} \mid q_i \in \text{children}(q')\}$ ,  $\text{Startmax} = \max\{C_{qi}.\text{start} \mid q_i \in \text{children}(q')\}$ . For any element  $C'_{q'}$ , which is  $C_q$  or any element after  $C_q$  in the stream of  $q'$ ,  $C'_{q'}$  is a quasi-potential element of  $q'$  if  $C'_{q'}.\text{start} < \text{Startmin}$  and  $C'_{q'}.\text{end} > \text{Startmax}$ . In addition, current elements of leaf nodes in stream  $q$  are also quasi-potential elements.*

**Definition 4(Potential Element).** *If  $q$  has a solution extension and  $q$  is the root of query  $Q$ , then quasi-potential elements of nodes in  $q$  are potential elements.*

**Definition 5(Minimal Leaf Element).** *Minimal leaf element,  $C_{L_{\min}}$ , is the leaf element whose start value is minimal among the current elements of all the leaf nodes in  $Q$ , that is,  $L_{\min} = \min \arg_{L_i} \{C_{Li}.\text{start} \mid L_i \text{ is any leaf node of } Q\}$ .*

**Example.** In Fig.1, when each cursor points to the first element of each stream,  $s1$  has a solution extension. As  $s$  has a solution extension,  $\text{Startmin} = \min\{C_p.\text{start}, C_t.\text{start}\} = C_t.\text{start}$ ,  $\text{Startmax} = \max\{C_p.\text{start}, C_t.\text{start}\} = C_p.\text{start}$ ,  $s1.\text{start} < \text{Startmin}$  and  $s1.\text{end} > \text{Startmax}$ , so  $s1$  is a quasi-potential element. In the same way,  $s2, s3$  are quasi-potential elements of  $s$ .  $t1, p1$  are quasi-potential elements of  $t, p$ , and  $k1, f1$  are quasi-potential elements of  $k, f$ . As node  $s$  is the root of  $Q1$ , so these elements are potential elements.  $k1$  is the current minimal leaf element.  $(s1, t1, k1, p1, f1)$  is a match of  $//s//t//k//p//f$ , and it is a solution of  $Q1$ .  $(s1, t1, k1)$  is a partial solution, where the path from the root to the leaf node,  $k$ , is  $//s//t//k$ .

According to above concepts, we can deduce that if a node has a solution extension, then all of its descendants in the query also have solution extensions, and Lemma 1 guarantees its correctness. While Lemma 2 assures that the potential elements in one non-leaf node are self-nested, and they should be pushed into corresponding stacks together directly according to Corollary 1, without calling subroutine *getNext* again and again to check whether their descendant nodes have solution extensions. Accordingly, Lemma 2 and Corollary 1 can avoid the self-nested suboptimal.

**Lemma 1.** *Suppose the root of query  $Q$  has a solution extension, then for any node  $q$  in  $Q$ ,  $q$  also has a solution extension.*

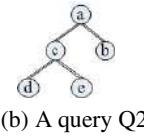
**Lemma 2.** Potential elements of the same non-leaf node  $q$  in  $Q$  are self-nested, that is, for potential elements  $e_i$  and  $e_j$  of  $q$ ,  $e_i$  is an ancestor of  $e_j$ , or vice versa.

**Corollary 1.** For any non-leaf nodes  $q_i$  and  $q_j$  in  $Q$ , and  $e_{qi}$ ,  $e_{qj}$  are potential elements of  $q_i$  and  $q_j$  respectively. If  $q_i$  is an ancestor of  $q_j$ , then  $e_{qi}$  must be an ancestor of  $e_{qj}$ .

**Example.** In Fig.2,  $a2$  has a solution extension, and  $a2, a3$  are quasi-potential elements of node  $a$ .  $b3, c2, d2, e2$  are quasi-potential elements of node  $b, c, d, e$  respectively. As  $a$  is the root of  $Q_2$ , so  $a2(a3)$ ,  $b3, c2, d2, e2$  are potential elements, and they compose a solution.  $a2$  and  $a3$  are self-nested according to Lemma 2. As  $a2$  and  $a3$  are both potential elements of  $a$ , so they are ancestors of  $b3, c2, d2, e2$  according to Corollary 1.

|                                            |                                                                    |                                                                                                                                                                   |                                                                                     |
|--------------------------------------------|--------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| $\frac{a1}{\underline{b1} \underline{c1}}$ | $\frac{}{\underline{b3} \underline{b4}} \frac{a3}{\underline{c2}}$ | $\frac{a2}{\underline{d3} \underline{e3}} \frac{c3}{\underline{d4} \underline{e4}} \frac{c4}{\underline{e4}} \frac{c5}{\underline{e4}} \frac{b5}{\underline{e4}}$ | $\frac{a4}{\underline{c6} \underline{c5}} \frac{b6}{\underline{d5} \underline{e5}}$ |
|--------------------------------------------|--------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|

(a) An element set

(b) A query  $Q_2$ **Fig. 2.** An XML element set and a query  $Q_2$ 

Potential elements are very crucial for answering twig patterns and Lemma 3 assures potential elements must contribute to final solutions and can not be discarded. Thus, after locating the match of  $q$  in  $Q$  on  $D$ , all the potential elements of non-leaf nodes should be pushed into corresponding stacks. Accordingly the self-nested suboptimal is avoided.

**Lemma 3.** Potential elements must contribute to some final solutions.

**Lemma 4.** Suppose node  $q$  (except for the root) in  $Q$  has a solution extension. If  $\exists e_{pq}$  an potential element of  $parent(q)$ , which is an ancestor of  $C_q$ , then quasi-potential elements of nodes in  $q$  are potential elements; otherwise, these quasi-potential elements are not potential elements.

We can distinguish which quasi-potential elements are potential elements and which are not according to Lemma 4. Once there is a potential element  $e_{pq}$  in  $parent(q)$ , which is an ancestor of  $C_q$ , the quasi-potential elements of nodes in  $q$  are potential elements and must contribute to solutions; otherwise, these quasi-potential elements are not potential elements and can be discarded. Since potential elements must contribute to final solutions according to Lemma 3, once locating the match of any node  $q$  in  $Q$ , we should push all the potential elements into corresponding stacks. Subsequently, we introduce Rule 1, which describes how to push potential elements into stacks, and Lemma 5 guarantees that the elements pushed into stacks via Rule 1 must be potential elements.

**Rule 1.** Suppose a non-leaf node  $q$  in  $Q$  has a solution extension,

- i) if  $q$  is the root, potential elements of any non-leaf node  $q'$  in  $q$  are pushed into  $S_{q'}$ .
- ii) Otherwise, only if element,  $S_{parent(q).bottom()}$ , is an ancestor of  $C_q$ , quasi-potential elements of each non-leaf node  $q'$  in  $q$  are pushed into  $S_{q'}$ .

**Lemma 5.** *The elements pushed into stacks via Rule1 must be potential elements.*

**Example.** In Fig.2, as  $c3$  has a solution extension, so  $c3$  is a quasi-potential element of  $c$ . Suppose  $Sa=\{a2,a3\}$ . As  $c3$  is not a descendant of  $a3$ , so  $c3$  is not a potential element for  $a3$ , but it is a potential element for  $a2$  according to Lemma 4.  $c3$  will be pushed into  $Sc$  according to Rule 1. If there is no potential element of node  $a$ , which is an ancestor of  $c3$ ,  $c3$  will not be a potential element. While  $c6$  has a solution extension and  $Sa=\{a2\}$ , as  $c6$  is not a descendant of  $a2$ , so  $c6$  is not a potential element of  $a2$  according to Lemma 4, thus  $c6$  will not be pushed into  $Sc$  immediately.

In this way, we first locate the match of a sub-query  $q$  in  $Q$  in root-to-leaf way, then push all the potential elements of nodes in  $q$  into corresponding stacks, and finally detect all the partial solutions through selecting current minimal leaf element,  $C_{Lmin}$ . Once  $S_{parent(Lmin)}$  is empty, the next match of a certain node should be located, but selecting which node is very crucial, therefore we introduce the notion of key node and Rule 2 to address this issue.

**Definition 6(Key Node).** Suppose  $q$  is the parent node of  $L_{min}$ ,  $C_{Lmin}$  is the current minimal leaf element. If  $S_q$  is empty, key node ( $k_q$ ) will be defined, which satisfies:

- i) if  $q$  is the root, then  $k_q=q$ ;
- ii) else if there is at least one element, which is an ancestor of  $C_{Lmin}$  in  $S_{parent(q)}$ , that is,  $S_{parent(q)}.bottom().end > C_{Lmin}.start$ , then  $k_q=q$ ;
- iii) else,  $k_q=q'$ ,  $q'$  is an ancestor node of  $q$ , and  $q'$  satisfies:
  - a)  $\exists e_{q'}$ , which is an ancestor of  $C_{Lmin}$  in  $S_{parent(q')}$ , but  $\neg \exists e_q$  which is an ancestor of  $C_{Lmin}$  in  $S_{q'}$  i.e.  $S_{parent(q)}.bottom().end > C_{Lmin}.start$ ,  $S_{q'}.bottom().end < C_{Lmin}.start$ ; or
  - b)  $q'$  is the root and  $\neg \exists e_{q'}$  which is an ancestor of  $C_{Lmin}$  in  $S_{q'}$ , i.e.,  $S_{q'}.bottom().end < C_{Lmin}.start$ .

**Rule 2.** Minimal leaf element,  $C_{Lmin}$ , is first selected to discover partial solutions,

- i) if after popping all the elements, which are not ancestors (or parents) of  $C_{Lmin}$  from  $S_{parent(Lmin)}$ , each stack from the root to  $parent(L_{min})$  is not empty, then  $(es_1, \dots, es_k, C_{Lmin})$  is a partial solution, where  $s_1, \dots, s_k$  are the stacks of nodes on the path from  $parent(L_{min})$  to the root, and  $es_i$  is any element in  $s_i$  ( $1 \leq i \leq k$ ).
- ii) Otherwise, the key node ( $k_q$ ) is detected from  $L_{min}$  to the root, and the match of  $k_q$  is located. All the elements are popped from  $S_{q'}$ , where  $q'$  is any non-leaf node in  $k_q$ .

In Rule 2, if  $S_{parent(C_{Lmin})}$  is not empty, the partial solutions will be outputted according to i); otherwise the next match of a certain node need to be located, but selecting which node to match is very important, hence key node is defined. If selecting key node to match, many useless computations will be avoided. This is because, current potential elements of nodes in sub-query  $k_q$  are processed already, and they will not contribute to any solution in the future, but elements in its ancestor stacks may contribute to solutions in the future. In Definition 6, i) means that if  $q(parent(L_{min}))$  is the root,  $Cq$  may be a potential element, thus  $q$  is selected as the key node; ii) means that  $C_{Lmin}$  is a descendant of an element in  $S_{parent(q)}$ , that is, there may be a potential element which is an ancestor of  $C_{Lmin}$  in  $S_{parent(q)}$ , and  $Cq$  may be a

potential element in the future, therefore,  $q$  is selected as key node; iii) means that current potential elements of  $q'$  and  $q''$ 's descendants have been processed, but elements of  $q'$ 's parent(if any) may contribute to solutions with the new quasi-potential elements of  $q'$  in the future, thus  $q'$  is selected as key node.

In addition, in Rule 2, once selecting a minimal leaf element, we only need to check whether there is an ancestor of this minimal leaf element in its parent stack, but do not need to call subroutine *getNext* from the root many times. We always select the minimal leaf element,  $C_{L_{min}}$ , to locate a partial solution according to Rule 2, which avoids checking whether  $L_{min}$ 's following siblings and  $L_{min}$ 's following siblings' descendants have solution extensions. Accordingly, we can avoid order suboptimal through Rule 2.

To check whether the minimal leaf element will contribute to partial solutions, we present Lemma 6. If  $L_{min}$  is a potential element, the partial solution that contains it will be outputted as i); otherwise, all the elements of nodes in  $k_q$  will be popped, which have ever been potential elements but have already been processed, from corresponding stacks as ii). At the same time, it assures the correctness of Rule 2.

**Lemma 6.** *The elements that are not ancestors of  $C_{L_{min}}$  in  $S_{parent(L_{min})}$  cannot contribute to partial solutions in the future. After popping these elements from  $S_{parent(L_{min})}$ ,*

- i) *if  $S_{parent(L_{min})}$  is still not empty, then  $(es_1, \dots, es_k, C_{L_{min}})$  is a partial solution, where  $s_1, \dots, s_k$  are the stacks of nodes on the path from  $parent(L_{min})$  to the root, and  $es_i$  is any element in  $s_i$  ( $1 \leq i \leq k$ ).*
- ii) *Otherwise, all the elements in  $S_{q_i}$  will not contribute to final solutions in the future and should be popped, where  $q_i$  is any descendant node (if any) of  $k_q$ .*

Rule 1 assures that potential elements are always pushed into corresponding stacks from-root-to-leaf, and self-nested suboptimal can be avoided. Rule 2 always selects the minimal leaf element,  $C_{L_{min}}$ , to discover partial solutions, and once  $S_{parent(L_{min})}$  is empty, the key node will be detected from-leaf-to-root and the next solution extension of it will be located. Once detecting  $C_q$  pointing to *nil*, Rule 3 avoids stream null suboptimal through locating  $C_{q'}$  to *nil*. Rule 1-3 together can avoid those three suboptimal. We in the next section propose an efficient twig join algorithm, and incorporate these three rules into it.

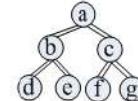
**Rule 3.** *Suppose key node  $k_q$  has no solution extension,  $q'$  is any node in  $k_q$  or an ancestor (if any) of  $k_q$ . Elements in stream  $q'$  will not be potential elements, and thus  $C_{q'}$  should be located to *nil* (beyond the last element of stream  $q'$ ).*

$k_q$  having no solution extension means that elements of nodes in  $k_q$  will not compose a match of  $k_q$ . Thus, once detecting one cursor  $C_{q_i}$  pointing to *nil*, where  $q_i$  is any node in  $k_q$ , we need not scan the elements of nodes in  $k_q$ . Rule 3 is employed to skip these elements, and its correctness is guaranteed by Lemma 7 and Lemma 8.

**Lemma 7.** *Suppose  $q$  is a node (except for the root) in  $Q$ , and  $C_q$  points to *nil*.  $\forall A_q$  which is an ancestor node of  $q$ ,  $C_{A_q}$  or elements after it in the stream of  $A_q$  will not be potential elements.*

**Lemma 8.** *Locating  $C_{q'}$  to *nil* will not miss any final solutions through Rule 3.*

| $a1$             |                  |                  |                  | $a2$             |                  |                  |                  |
|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| $b1$             | $c1$             | $d1$             | $e1$             | $f1$             | $g1$             | $d2$             | $e2$             |
| $\underline{d1}$ | $\underline{e1}$ | $\underline{f1}$ | $\underline{g1}$ | $\underline{f2}$ | $\underline{g2}$ | $\underline{d3}$ | $\underline{e3}$ |



**Fig. 3.** An element set and a query Q3

**Example.** In Fig.3, when each cursor points to the first element of each stream,  $a1$  has a solution extension, and  $a1, b1, c1$  are pushed into  $Sa, Sb, Sc$  respectively. When  $C_{Lmin}=d2$ ,  $b1$  is popped from  $Sb$ , and  $k_q$  is  $b$ . As  $Cb=nil$ , so the cursors of its sub-nodes,  $Cd, Ce$  and its ancestor  $Ca$  should be located to nil. However,  $Cc$  can not be located to nil, because  $Sa$  is not empty, and some elements in stream  $c$  may be potential elements, such as  $c2$ . While  $C_{Lmin}=f3$ ,  $a1$  is popped from  $Sa$ ,  $c2$  from  $Sc$ , and  $k_q$  is  $a$ . As  $Sa=\{\}$ , so  $Cc, Cf, Cg$  are located to nil. Accordingly, elements  $e2, d3, e3, c3, g3, c4, f4, g4$  are skipped.

Rule 1 assures that potential elements are always pushed into corresponding stacks from-root-to-leaf and thus self-nested suboptimal can be avoided. Rule 2 always selects the minimal leaf element,  $C_{Lmin}$ , to discover partial solutions. In addition, once detecting  $Cq$  pointing to  $nil$ , Rule 3 avoids stream null suboptimal through locating some cursors to  $nil$ . Rule 1-3 together avoid those three suboptimal. We in the next section propose an efficient twig join algorithm and incorporate the three rules into it.

## 4.2 The TJEssential Algorithm

TJEssential operates two phases, as follows:

1. *Output partial solutions.* TJEssential discovers and outputs partial solutions according to the three Rules in section 4.1. It first locates the match of a sub-query  $q$  of  $Q$  and pushes the potential elements of non-leaf nodes in  $q$  into corresponding stacks through Rule 1, and then it selects the current minimal leaf element to output partial solutions: if this minimal leaf element can contribute to partial solutions together with potential elements in current stacks, it outputs this partial solution and selects the next minimal leaf element as Rule 2; otherwise, it detects the key node as Definition 6. If this key node has a solution extension, TJEssential locates the match of it; otherwise locates all the cursors of this key node's ancestors and descendants to  $nil$  as Rule 3.
2. *Merge partial solutions to final solutions.* All the partial solutions in the first phase are merged to produce the final solutions to the whole twig pattern.

The main difference between our algorithm and existing proposals(e.g. TSGeneric) is that, to output a partial solution, existing proposals will recursively call its subroutine *getNext* many times, i.e., TSGeneric will check whether the nodes in  $Q$  have solution extensions many times and involves unnecessary computations, but TJEssential won't. Once detecting a match of a sub-query of  $Q$ , TJEssential pushes all the potential elements of non-leaf nodes in this sub-query into corresponding stacks. Moreover, it selects the key node to locate the next match, but does not always locate a match of the root as existing studies. Accordingly, TJEssential avoids many unnecessary computations.

According to the three Rules in 4.1, we demonstrate our algorithm, TJEssential. It first locates the match of a node in Q through calling *getMatch* in line 1, and then discovers and outputs all the partial solutions according to Rule 2 in lines 2-10, finally it sticks the partial solutions in line 11. TJEssential always selects the minimal leaf element,  $C_{L_{min}}$  to discover partial solutions in line 3, and if  $L_{min}$  is not the root, pops the elements from  $S_{parent(L_{min})}$  which are not ancestor of  $C_{L_{min}}$  in line 4-5; otherwise, if  $L_{min}$  is the root or  $S_{parent(L_{min})}$  is not empty, TJEssential outputs partial solutions and advances  $C_{L_{min}}$  in line 6-8; otherwise, finds the next key node  $k_q$  from the leaf to the root and locates the match of  $k_q$  in line 9.

Algorithm 1 *TJEssential()*

```

1. getMatch(root);
2. while not end(root) do
3. $L_{min} = \text{minarg}_{Li}(C_{Li}.start)$;
4. if not isRoot(L_{min}) then
5. cleanStack($S_{parent(L_{min})}, C_{L_{min}}$);
6. if isRoot(L_{min}) or (not empty($S_{parent(L_{min})}$)) then
7. outputPartialSolutions($C_{L_{min}}$);
8. $C_{L_{min}} > \text{advance}()$;
9. else getMatch(parent(L_{min}));
10. end while
11. mergeAllPartialSolutions();

```

*Function end(q)*

If all the leaf streams in sub-query q are empty then return true; else return false.

*Procedure cleanStack(Sq, Cp)*

Pop all the elements from Sq that are not ancestors of Cp;

*Procedure mergeAllPartialSolutions()*

Merge the partial solutions to final solutions;

*Procedure outputPartialSolutions(C<sub>L<sub>min</sub></sub>)*

Output partial solutions that contain  $C_{L_{min}}$ ;

Algorithm 2 *getMatch*(q)

**Input:** node q in the query Q.  
 {Locate the next match of the key node}  
 1.  $q = \text{getKeyNode}(q)$ ;  
 2. **if** (**not** *LocateExtension*(q))  
 3. *Locate2End*(q);  
 4. return;  
 5. **if** *isRoot*(q) **and not** *isLeaf*(q) **then**  
 6. *Push2Stack*(q);  
 6. **else if**  $C_q.start < S_{parent(q)}.bottom().end$  **then**  
 7. *cleanStack*( $S_{parent(q)}, C_q$ );  
 8. *Push2Stack*(q);  
 9. **else** return;

*Function getKeyNode(q)*

Return the key node according to Definition 6.

*Function LocateExtension(q)*

1. **if** q has a solution extension **then**  
 2. locate the match of q; return true;  
 3. **else** return false.

*Procedure Locate2End(q)*

Locate  $C_q'$  to nil via Rule 3, where  $q'$  is any ancestor or descendant of q;

*Procedure Push2Stack(q)*

Pop the elements from the stacks of nodes in q, and push the quasi-potential elements into corresponding stacks.

Algorithm *getMatch* is used to detect the next key node  $k_q$  from-leaf-to-root and then locate the next match of  $k_q$ . It detects the key node in line 1, where *getKeyNode* is used to get the key node according to Definition 6, then calls *LocateExtension* to locate the match of  $k_q$  in line 2, where *LocateExtension*(q) is used to locate the match of a sub-query q as [JWL<sup>+</sup>03], but the difference is that when q has no solution extension, it returns false. If it returns false, *getMatch* calls *Locate2End*(q) to locate the cursors of certain nodes to the end of their streams according to Rule 3 in line 3. Otherwise, if  $k_q$  is the root, it pushes potential elements into corresponding stacks by calling *push2Stack* in line 5, where *push2Stack*(q) is used to push potential elements of non-leaf nodes in q into corresponding stacks recursively according to Rule 1. If there are some elements in  $S_{parent(q)}$  that are ancestors of  $C_q$  ( $C_q.start < S_{parent(q)}.bottom().end$ ), it will call *push2Stack*(q) to push potential elements of nodes in q into corresponding stacks in lines 6-8, and TJEssential will continue to

discover partial solutions of this solution extension; otherwise, it returns to TJEssential to deal with the next minimal leaf element in line 9.

**Theorem 1.** *Given a twig query  $Q$  and an XML database  $D$ , the TJEssential algorithm correctly returns all the answers for  $Q$  on  $D$ .*

### 4.3 The TJEssential\* Algorithm

TJEssential is optimal when the twig pattern only involves A-D relationships. To efficiently support P-C relationships, we propose TJEssential\*, which introduces Tag+Level stream and PPS stream into TJEssential. Therefore, TJEssential\*(Tag+Level) is optimal when the twig pattern involves A-D only or P-C only relationships, and TJEssential\*(PPS) is optimal for A-D only, P-C only and one branch node only twig patterns assuming there is no repetitive tag in the twig patterns. TJEssential\* is similar to TJEssential, and the only difference is that, the former employs tag streams, but the latter employs Tag+Level or PPS streams. In addition, our three optimization rules can be incorporated into Twig<sup>2</sup>Stack.

## 5 Experiment

In this section, we present the experiments conducted to evaluate the efficiency of various algorithms and report some of the results obtained.

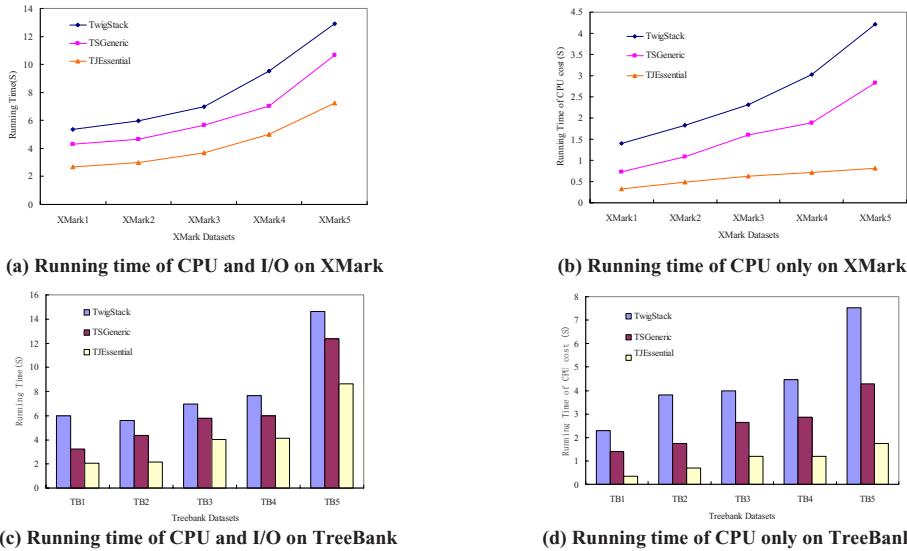
We compared TJEssential with TwigStack, TSGeneric when the twig pattern only involved A-D relationships, while if the twig pattern involved A-D and P-C relationships, TJEssential\* was more efficient and could avoid some intermediate results as iTwigJoin. Subsequently, we compared TJEssential\* (Tag+Level) with iTwigJoin(Tag+Level), but iTwigJoin(PPS) and TJEssential\* (PPS) were omitted, because both of them involved too many streams and induced inefficiency, especially when the depth of an XML document was too deep. All the algorithms were coded using Microsoft Visual C++ 6.0, and experiments were conducted on an AMD XP 2600+ PC with 1G RAM, running Windows 2000 server. We used real-world [TreeBank] and synthetic[XMark] datasets for our experiments: (1)XMark, which is synthetic and generated by an XML data generator. It has many repetitive structures and fewer recursions. The benchmark data was generated with SF(scale factor)=1, and the raw text file was 113MB; (2) TreeBank(TB), which is obtained from the University of Washington XML repository and the text file is 82MB. The DTD of Treebank is deep recursive, and the deep recursive structure of this data makes it ideal for experiments of twig pattern matching algorithms.

We selected eight queries for XMark, TB respectively as shown in Table 2. There are some branching nodes in the first five XMark queries (XMark 1-5), thus TwigStack and TSGeneric involve much order and stream null suboptimal on these five queries. For the first five TB queries(TB 1-5), there are some nodes, i.e., NP, PP, which are self-nested, therefore TwigStack and TSGeneric involve much self-nested sub-optimal on these five queries. To compare TJEssential\* with iTwigJoin, we devised several queries XMark 6-8 and TB 6-8, which contained both “//” and “/”.

**Table 2.** Queries used in our experiments

|        |                                                                                                                         |                                          |
|--------|-------------------------------------------------------------------------------------------------------------------------|------------------------------------------|
| XMark1 | //person[//profile[//age][//interest][//education][//gender][//business]][//address]/emailaddress                       |                                          |
| XMark2 | //person[//emailaddress][//homepage][//name][//address][//country][//city]                                              |                                          |
| XMark3 | //site[//person[//homepage][//emailaddress]][//open_auction[//bidder][//reserve]][//closed_auction[//annotation]]/price |                                          |
| XMark4 | //closed_auction[//annotation[//description]][//price][//date][//buyer][//seller]                                       |                                          |
| XMark5 | //open_auction[//bidder[//personref][//time][//date]][//quantity][//reserve][//current]                                 |                                          |
| TB1    | //S//VP//PP[//NP//VBN]//IN                                                                                              | TB3 //S[//VP[//NN][//VBD]]//NP[//IN]//DT |
| TB2    | //S//NP[//PP//TO][//VP[//NONE_]/JJ]                                                                                     | TB4 //S[//NP][//NONE_]/VP//PP[//IN]//DT  |
| TB5    | //S[//NP[//DT][//NN]]//PP[//IN]/NN                                                                                      |                                          |
| XMark6 | //text[//bold][//keyword]//emph                                                                                         | TB6 //S[//VP[//NN][//VBD]]//NP[//IN]//DT |
| XMark7 | //listitem[//bold]//text//emph                                                                                          | TB7 //S[//NP[//NONE_]/VP//PP[//IN]]//DT  |
| XMark8 | //listitem[//bold]//text[//emph]//keyword                                                                               | TB8 //S[//NP[//DT][//NN]]//PP[//IN]/NN   |

To compare these algorithms, we introduce a metric, Improved Ratio (IR), and  $IR_{X,Y} = (T_X - T_Y)/T_X$ , where  $T_X$  and  $T_Y$  are the running times for algorithms X and Y respectively. TJEssential mainly improves CPU cost of existing algorithms, thus we also compare running time of CPU cost only (without I/O cost), and employ another metric  $IR'_{X,Y} = (T'_X - T'_Y)/T'_X$ , where  $T'_X$  and  $T'_Y$  are running time of CPU cost only for algorithms X and Y.

**Fig. 4.** Queries only involving “//”

On XMark,  $IR_{\text{TwigStack}, \text{TJEssential}}$  is more than 30% and  $IR_{\text{TSGeneric}, \text{TJEssential}}$  is more than 20% when considering running time of both CPU and I/O cost as shown in Fig.4(a). If only considering running time of CPU without I/O cost,  $IR'_{\text{TwigStack}, \text{TJEssential}}$  is at least 50% and  $IR'_{\text{TSGeneric}, \text{TJEssential}}$  is at least 35% as shown in Fig.4(b). Especially,  $IR'_{\text{TwigStack}, \text{TJEssential}}$  exceeds 68% and  $IR'_{\text{TSGeneric}, \text{TJEssential}}$  exceeds 41% on XMark1.

On Treebank,  $IR_{\text{TwigStack}, \text{TJEssential}}$  is at least 40% and  $IR_{\text{TSGeneric}, \text{TJEssential}}$  is at least 30% as illustrated in Fig.4(c). Since Treebank is deep recursive, TJEssential avoids

self-nested, order and stream null suboptimal together. Especially,  $IR'_{TwigStack,TJEssential}$  exceeds 85% and  $IR'_{TSGeneric,TJEssential}$  exceeds 75% on TB1 as shown in Fig.4(d).

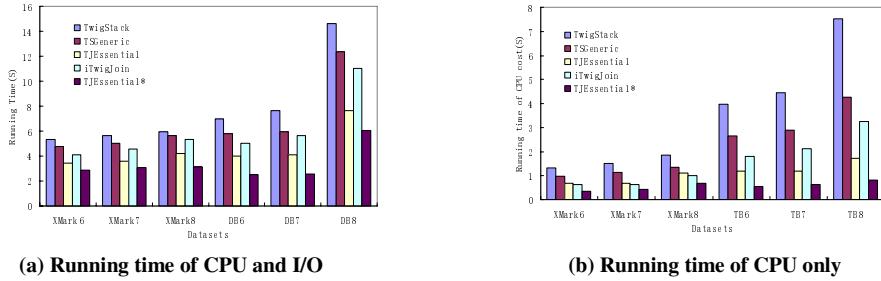


Fig. 5. Queries involving “//” and “/”

To compare  $TJEssential^*$  (Tag+Level) with iTwigJoin (Tag+Level), we employ six queries: XMark 6-8 and TB 6-8.  $TJEssential^*$  is more efficient than TwigStack, TSGeneric and iTwigJoin when the query contains both “//” and “/”.  $IR_{TwigStack,TJEssential^*}$ ,  $IR_{iTwigJoin,TJEssential^*}$  and  $IR_{TSGeneric,TJEssential^*}$  are more than 50% on TB6,TB7 as shown in Fig.5(a). Especially,  $IR'_{TwigStack,TJEssential^*}$ ,  $IR'_{TSGeneric,TJEssential^*}$  and  $IR'_{iTwigJoin,TJEssential^*}$  reach 89%, 80%, 74% respectively on TB8 as shown in Fig.5(b). In conclusion,  $TJEssential$  and  $TJEssential^*$  solve the disadvantages of existing algorithms that involve too many unnecessary computations, and explore the potential benefit of CPU cost on self-nested, order, stream null suboptimal.

## 6 Conclusion

This paper addresses the problem that there are unnecessary computations of existing holistic twig join algorithms. We first propose three optimization rules to avoid self-nested, order and stream null suboptimal of existing studies, and then present an effective twig join algorithm,  $TJEssential$ , to explore potential benefit of CPU cost. Incorporated with these three optimizations,  $TJEssential$  can speed up answering twig patterns through discovering the solutions in leaf-to-root in accordance with root-to-leaf way. Experimental results show that our approach achieves high efficiency and outperforms existing proposals.

## Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grant No.60573094, the National Grand Fundamental Research 973 Program of China under Grant No.2006CB303103, the National High Technology Development 863 Program of China under Grant No.2006AA01A101, Tsinghua Basic Research Foundation under Grant No. JCqn2005022, and Zhejiang Natural Science Foundation under Grant No. Y105230.

## References

- [AJK<sup>02</sup>] S.Al-Khalifa, H.V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava and Y. Wu. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In ICDE, pages 141-152, 2002.
- [BBC<sup>02</sup>] A. Berglund, S. Boag, D. Chamberlin et al. XML path language 2.0. Technical report, W3C, 2002.
- [BCF<sup>02</sup>] S. Boag, D.Chamberlin, M.Fernandez et al. XQuery 1.0: An XML query language. W3C, 2002.
- [BKS02] N. Bruno, N. Koudas et al. Holistic Twig Joins: Optimal XML Pattern Matching. In SIGMOD, 2002.
- [CLC04] T. Chen, T. W. Ling, and C. Y. Chan. Prefix path streaming: a new clustering method for optimal XML twig pattern matching. In DEXA, pages 801–811, 2004.
- [CLL05] T. Chen, J. Lu, and T.W. Ling. On Boosting Holism In XML Twig Pattern Matching Using Structural Indexing Techniques. In SIGMOD, 2005.
- [CLT<sup>06</sup>] Songting Chen, Hua-Gang Li, Junichi Tatenuma, Wang-Pin Hsiung, Divyakant Agrawal and K. Candan Twig<sup>2</sup>Stack: Bottom-up Processing of Generalized-Tree-Pattern Queries over XML Documents. In VLDB, pages 283-294, 2006.
- [CMW03] B. Choi, M. Mahoui et al. On the Optimality of Holistic Algorithms for Twig Queries. In DEXA, 2003.
- [CVZ<sup>02</sup>] S.-Y.Chien, Z. Vagena et al. Efficient Structural Joins on Indexed XML Documents. In VLDB, 2002.
- [FK99] D. Florescu,D. Kossmann. Storing and querying XML data using an RDBMS. IEEE Data Eng., 1999.
- [G02] T. Grust. Accelerating XPath Location Steps. In SIGMOD, pages 109-120, 2002.
- [JLWO03] H.F. Jiang, H.J. Lu et al. XR-Tree:Indexing XML Data for Efficient Structural Joins. In ICDE, 2003.
- [JWL<sup>03</sup>] H.F. Jiang, W. Wang, H.J. Lu et al. Holistic Twig Joins on Indexed XML Documents. In VLDB, 2003.
- [JLW04] H.F. Jiang, H.J. Lu et al. Efficient Processing of Twig Queries with OR-Predicates. In SIGMOD, 2004.
- [LCL04] Jiaheng Lu, Ting Chen and Tok Wang Ling. Efficient Processing of XML Twig Patterns with Parent Child Edges: A Look-ahead Approach. In CIKM, pages 533-542,2004.
- [LLC<sup>05</sup>] Jiaheng Lu, Tok Wang Ling, Chee-Yong Chan and Ting Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. In VLDB, pages 193-204, 2005.
- [LM01] Q. Li and B. Moon. Indexing and Quering XML Data for Regular Path Expressions. In VLDB, 2001.
- [MHH06] Christian Mathis, Theo Härdter and Michael Haustein. Locking-Aware Structural Join Operators for XML Query Processing. In SIGMOD, 2006.
- [STZ<sup>99</sup>] J. Shanmugasundaram, K. Tufte, C. Zhang, H. Gang, D. J. DeWitt, and J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In VLDB, pages 302-314, 1999.
- [TVB<sup>02</sup>] I. Tatarinov, S. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In SIGMOD, pages 204-215, 2002.

- [WPJ03] Y. Wu, J. Patel and H. Jagadish. Structural join order selection for XML query optimization. In ICDE, 2003.
- [ZND<sup>+</sup>01] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. On Supporting Containment Queries in Relational Database Management Systems. In SIGMOD, pages 425-436, 2001.
- [TreeBank] University of Washington XML Repository. <http://www.cs.washington.edu/research/xmldatasets/>.
- [XMark] <http://monetdb.cwi.nl/xml>

# **TwigList: Make Twig Pattern Matching Fast**

Lu Qin, Jeffrey Xu Yu, and Bolin Ding

The Chinese University of Hong Kong, China  
`{lqin, yu, blding}@se.cuhk.edu.hk`

**Abstract.** Twig pattern matching problem has been widely studied in recent years. Give an XML tree  $\mathcal{T}$ . A twig-pattern matching query,  $Q$ , represented as a query tree, is to find all the occurrences of such twig pattern in  $\mathcal{T}$ . Previous works like *HolisticTwig* and *TJFast* decomposed the twig pattern into single paths from root to leaves, and merged all the occurrences of such path-patterns to find the occurrences of the twig-pattern matching query,  $Q$ . Their techniques can effectively prune impossible path-patterns to avoid producing a large amount of intermediate results. But they still need to merge path-patterns which occurs high computational cost. Recently, *Twig<sup>2</sup>Stack* was proposed to overcome this problem using hierarchical-stacks to further reduce the merging cost. But, due to the complex hierarchical-stacks *Twig<sup>2</sup>Stack* used, *Twig<sup>2</sup>Stack* may end up many random accesses in memory, and need to load the whole XML tree into memory in the worst case. In this paper, we propose a new algorithm, called *TwigList*, which uses simple lists. Both time and space complexity of our algorithm are linear with respect to the total number of pattern occurrences and the size of XML tree. In addition, our algorithm can be easily modified as an external algorithm. We conducted extensive experimental studies using large benchmark and real datasets. Our algorithm significantly outperforms the up-to-date algorithm.

## 1 Introduction

The Extensible Markup Language (XML) is an emerging standard for data representation and exchange on the Internet. Pattern matching is one of the most important types of XML queries to retrieve information from an XML document. Among many reported studies, Zhang et al. in [1] introduced the region encoding to process XML queries and proposed a multi-predicate merge join algorithm using inverted list. Al-Khalifa et al. in [2] proposed a stack-based algorithm which breaks the twig query into a set of binary components. The drawback of the early work is the large intermediate results generated by the algorithm. Bruno et al. in [3] used a holistic twig join algorithm *TwigStack* to avoid producing large intermediate results. Jiang et al. in [4] proposed an XML Region Tree (*XR-tree*) which is a dynamic external memory index structure specially designed for nested XML data. With *XR-tree*, they presented a *TSGeneric+* algorithm to effectively skip both ancestors and descendants that do not participate in a join. Lu et al. in [5] proposed *TwigStackList* to better handle twig queries with parent-child relationships. Lu et al. in [6] used a different labeling scheme called extended Dewey, and proposed a *TJFast* algorithm to access only leaf elements. However, all of the above algorithms can not avoid a large number of unnecessary path mergings as theoretically shown in [7]. Hence, Aghili et al. in [8] proposed a binary labeling algorithm using

the method of nearest common ancestor to reduce search space. However, this technique is efficient in the cases when the returned nodes are the leaf nodes in the twig query. Most recently, Chen et al. in [9] proposed a *Twig<sup>2</sup>Stack* algorithm which uses hierarchical-stacks instead of enumeration of path matches. *Twig<sup>2</sup>Stack* outperforms *TwigStack* and *TJFast*. But *Twig<sup>2</sup>Stack* may conduct many random accesses and may use a large memory space due to the complexity of hierarchical-stacks it uses.

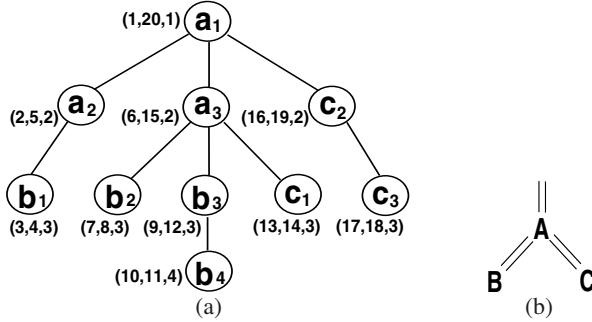
The main contribution of this paper is summarized below. We present a new algorithm, called *TwigList*, which is independently developed and shares similarity with *Twig<sup>2</sup>Stack* [9]. Our algorithm significantly outperforms *Twig<sup>2</sup>Stack*. The efficiency of our *TwigList* algorithm is achieved by using simple lists rather than the hierarchical-stacks used in *Twig<sup>2</sup>Stack* to reduce the computational cost. In addition, because of the simple list data structure and maximization of possible sequential scans used in our algorithm, we extend *TwigList* as an external algorithm, which still outperforms *Twig<sup>2</sup>Stack* using a 582MB XMark benchmark, a 337MB DBLP dataset, and a 84MB TreeBank dataset, as reported in our extensive experimental studies.

The remainder of this paper is organized as follows. Section 2 gives the problem of processing twig-pattern matching queries. Section 3 discusses two existing algorithms and outlines their problems. We give our new algorithm in Section 4. Experimental results are presented in Section 5. Finally, Section 6 concludes the paper.

## 2 Twig-Pattern Matching Queries

An XML document can be modeled as a rooted, ordered, and node-labeled tree,  $\mathcal{T}$ , where a node represents an XML element, and an edge represents a parent/child relationship between elements in XML. For simplicity, in this work, a label of a node is a value that belongs to a type (tag-name). An example of an XML tree is shown in Fig. 1(a). In the XML tree, a node is associated with a value  $x_i$  which belongs to a type  $X$  (denoted  $x_i \in X$ ). For example, the root node has a value  $a_1$  that belongs to type  $A$ . The ordering among sibling nodes specifies a traversal order.

A twig-pattern matching query is a fragment of XPATH queries that can be represented as a query tree,  $Q(V, E)$ . Here,  $V = \{V_1, V_2, \dots, V_n\}$  is a set of nodes representing types. We let  $V_i$  denote both the  $i^{th}$  typed query node in  $Q$  and the set of the  $i^{th}$  typed elements in the XML tree  $\mathcal{T}$ ,  $E$  is a set of edges. An edge between two typed nodes, for example,  $A$  and  $D$ , is either associated with an XPATH axis operator  $//$  or  $/$  to represent  $A//D$  or  $A/D$ . Given an XML tree  $\mathcal{T}$ , the former is to retrieve all  $A$  and  $D$  typed elements that satisfy the ancestor/descendant relationships, and the latter is to retrieve all  $A$  and  $D$  typed elements that satisfy parent/child relationships. We call the former  $//$ -edge and the latter  $/$ -edge in short. As a special case, the root node has an incoming  $//$ - or  $/$ -edge to represent an XPATH query,  $//A$  or  $/A$ , suppose the root node is  $A$ -typed. The answer of a  $n$ -node query tree,  $Q(V, E)$ , against an XML tree  $\mathcal{T}$ , is a set of all  $n$ -ary tuples  $(v_1, v_2, \dots, v_n)$  in  $\mathcal{T}$ , for  $v_i \in V_i$  ( $1 \leq i \leq n$ ), that satisfy all the structural relationships imposed by  $Q$ . Consider an XPATH  $Q = //A[//C]/B$ . The query tree is illustrated in Fig. 1(b). When  $Q$  is issued against the XML tree (Fig. 1(a)), the answer includes  $(a_1, b_1, c_3)$  and  $(a_3, b_3, c_1)$ .



**Fig. 1.** An XML tree (a) and a query tree  $Q$  (b)

In this paper, we focus ourselves on efficient processing twig-pattern matching queries. For efficiently determining ancestor/descendant relationships among nodes in an XML tree, a node  $u$  is encoded with a triple,  $(s_u, e_u, d_u)$ , where  $s_u$  and  $e_u$  together represent a region (starting/ending position), denoted  $\text{reg}(u)$ , and  $d_u$  represents the level of the node in the XML tree. (The root is at level 1.) With the region encoding (starting/ending position), a node  $u$  is an ancestor of a node  $v$  iff the  $\text{reg}(v) \subseteq \text{reg}(u)$  such that  $s_u < s_v \leq e_v < e_u$ , a node  $u$  is a parent of a node  $v$  iff  $\text{reg}(v) \subseteq \text{reg}(u)$  and  $d_v = d_u + 1$ , which implies that the node  $v$  is one level deeper than node  $u$ . For example, as shown in Fig. 1(a),  $b_2$  is a descendant of  $a_1$ , because the region of  $b_2$ ,  $(7, 8)$ , is contained in the region of  $a_1$   $(1, 20)$ . Also,  $b_2$  is not a child of  $a_1$  because the levels for  $b_2$  and  $a_1$  are 3 and 1.

### 3 Two Existing Algorithms: *TwigStack* and *Twig<sup>2</sup> Stack*

The twig-pattern matching query was first studied by Bruno, Koudas and Srivastava in [3]. A *TwigStack* algorithm was proposed to process a twig-pattern matching query,  $Q$ , in two steps. In the first step, in brief, a *PathStack* algorithm was proposed to efficiently process every query path in a given query tree. Consider the query  $Q = //A//C//B$  (Fig. 1(b)). There are two query paths,  $Q_{p_1} = //A//B$  and  $Q_{p_2} = //A//C$ . The *PathStack* algorithm finds answers for both of them using stacks. In the second step, *TwigStack* checks if the results for all the query paths can be merged to satisfy the structural relationships imposed by the given twig-pattern matching query. For *TwigStack*, the first step can be processed efficiently, but the second step consumes much time because it needs to process merging.

Below, in brief, we discuss the difficulties for *TwigStack* to reduce computational cost for merging in the second step after introducing *PathStack*. Consider a query path in the query tree  $Q_p = //V_1//V_2//\dots//V_n$ . A stack is created for every node  $V_i$ , denoted  $\text{stack}(V_i)$ . The whole query path is processed while traversing the given XML tree  $T$  following the preorder. When traversing a  $V_i$ -typed node  $v_i$  in XML tree  $T$  ( $v_i \in V_i$ ), *PathStack* pops up nodes that are not ancestors of  $v_i$  in  $\text{stack}(V_i)$  and  $\text{stack}(V_{i-1})$ , because they are no longer needed. Then *PathStack* pushes node  $v_i$  into  $\text{stack}(V_i)$ , iff

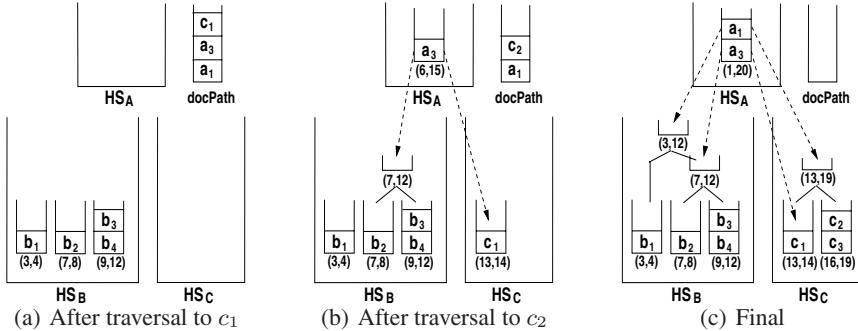
$stack(V_{i-1})$  is not empty. When  $v_i$  can be pushed into  $stack(V_i)$ , there is a pointer from  $v_i$  pointing to the top element in  $stack(V_{i-1})$ . Consider processing query path  $Q_{p_1} = //A//B$  against XML tree  $\mathcal{T}$  (Fig. 1(a)). There are two stacks  $stack(A)$  and  $stack(B)$ . Following preorder traversal,  $PathStack$  pushes  $a_1$  and  $a_2$  into  $stack(A)$ . When  $b_1$  is traversed, the top element of  $a_2$  in  $stack(A)$  is an ancestor of  $b_1$ , so  $b_1$  is pushed into  $stack(B)$ .  $PathStack$  will report  $(a_1, b_1)$  and  $(a_2, b_1)$  as result for the query path, because all the other elements in  $stack(A)$  are ancestors of the top element. Then,  $a_3$  is traversed, and  $PathStack$  will pop up  $a_2$  before pushing  $a_3$  into  $stack(A)$ , because  $a_2$  is not an ancestor of  $a_3$  and is not needed in the later processing. Similarly, when  $b_2$  is traversed,  $b_1$  is popped up. The merging process ensures the results satisfying the entire structural relationships. Reconsider  $Q = //A//C//B$  (Fig. 1(b)) against XML tree  $\mathcal{T}$  (Fig. 1(a)). Here,  $(a_3, b_2)$  satisfies  $Q_{p_1} = //A//B$ ,  $(a_1, c_2)$  satisfies  $Q_{p_2} = //A//C$ , but the two do not jointly satisfy  $Q = //A//C//B$ . The cost of merging is considerably high as processing  $n$  joins, if there are  $n$  query paths for a twig-pattern matching query.

It is worth noting that  $TwigStack$  cannot allow the same stack, say  $stack(A)$ , to be shared by two query paths  $Q_{p_1} = //A//B$  and  $Q_{p_2} = //A//C$ , and process twig-pattern matching queries without the merging step. It is because the sibling relationships cannot be easily maintained in the framework of  $TwigStack$ , and the push/popup, that are designed for each query path, cannot be used to control multiple paths (branches). Due to the different timing of push/popup, some answer may be missing.

In order to avoid the high cost in the step of merging, Chen et al. in [9] proposed a  $Twig^2 Stack$  algorithm which instead uses a hierarchical-stack, denoted  $HS_{V_i}$ , for each node, in query tree  $Q$ , to compactly maintain all twig-patterns for a twig-pattern matching query.

Consider a query tree  $Q(V, E)$  with  $n$  nodes ( $V = \{V_1, V_2, \dots, V_n\}$ ).  $Twig^2 Stack$  maintains  $n$  hierarchical-stacks  $HS_{V_i}$  for  $1 \leq i \leq n$ . Each  $HS_{V_i}$  maintains an ordered sequence of stack-trees,  $ST_1(V_i), ST_2(V_i), \dots$ , and a stack-tree,  $ST_j(V_i)$ , is an ordered tree of stacks. Each stack in the stack-tree contains zero or more document elements. The ancestor/descendant relationships are maintained by the stacks in the hierarchical-stacks. Suppose in an XML tree,  $u$  is an ancestor of  $v$ . If  $u$  and  $v$  have the same type, say  $V_i$ , in  $Twig^2 Stack$ , they may appear in the same stack. If so,  $v$  will be pushed into the stack before  $u$  is in  $HS_{V_i}$ . If  $u$  and  $v$  have different types,  $V_i$  and  $V_j$ , then  $u$  will be in one stack in  $HS_{V_i}$  and  $v$  will be in one stack in  $HS_{V_j}$  and there is a pointer from the stack in  $HS_{V_i}$  to the stack in  $HS_{V_j}$  to represent their ancestor/descendant relationship.

Take an example of processing  $Q = //A//C//B$  (Fig. 1(b)) against XML tree  $\mathcal{T}$  (Fig. 1(a)).  $Twig^2 Stack$  traverses  $\mathcal{T}$  in preorder:  $a_1, a_2, b_1, a_3, b_2, b_3, b_4, c_1, c_2$ , and  $c_3$ , and will push them into a special stack called  $docpath$  in such order. Initially,  $a_1$ ,  $a_2$  and  $b_1$  are pushed into the stack  $docpath$  in order. When  $Twig^2 Stack$  is about to push  $a_3$  into  $docpath$ , it finds that  $b_1$  is not an ancestor of  $a_3$  and therefore pops-up  $b_1$  from  $docpath$  and pushes  $b_1$  to the hierarchical-stack  $HS_B$ , and it then finds that  $a_2$  is not an ancestor of  $a_3$  either and therefore simply discards it (because  $a_2$  does not have any  $C$ -typed descendants now, and will not have any later). When  $Twig^2 Stack$  is about to push  $b_3$  into  $docpath$  after pushing  $b_2$  into  $docpath$ ,  $Twig^2 Stack$  finds that  $b_2$  is not  $b_3$ 's ancestor, it pops up  $b_2$  from  $docpath$  and pushes  $b_2$  into  $HS_B$ . Since  $b_2$  is not an ancestor of  $b_1$ , there will be two single-node stack-trees in  $HS_B$ . Fig. 2(a) shows



**Fig. 2.**  $\text{Twig}^2\text{Stack}$  for the query tree  $Q$  (Fig. 1(b)) against XML tree  $T$  (Fig. 1(a))

the *docpath*, the hierarchical-stacks after  $c_1$  is pushed into *docpath*. Fig. 2(b) shows the *docpath*, the hierarchical-stacks,  $HS_A$  and  $HS_B$  and  $HS_C$ , after  $c_2$  is pushed into *docpath*. Note: there are two stack-trees in  $HS_B$ . From  $a_3$  in  $HS_A$ , there is a pointer pointing to a subtree in  $HS_B$  indicating that it is an ancestor of  $b_2$ ,  $b_3$  and  $b_4$ ; also there is a pointer to  $HS_C$  indicating that  $a_3$  is an ancestor  $c_1$ . Fig. 2(c) shows the hierarchical-stacks after all XML tree nodes are pushed/popped-up into/from *docpath*. As can be seen from Fig. 2(c), all twig-patterns are maintained by the stacks in the hierarchical-stacks. After the hierarchical-stacks are constructed,  $\text{Twig}^2\text{Stack}$  enumerates the results in a bottom-top manner. For example, for  $a_1$ ,  $\text{Twig}^2\text{Stack}$  enumerates the stacks, and conduct Cartesian-product between  $a_1$  and  $\{b_1, b_2, b_3, b_4\}$  and  $\{c_1, c_2, c_3\}$ .

As shown in [9],  $\text{Twig}^2\text{Stack}$  is a linear-time (w.r.t. the number of nodes of  $T$ ) algorithm to construct the hierarchical-stacks, and is a linear-time (w.r.t. the total number of matchings) enumeration algorithm based on intermediate structures maintained in the hierarchical-stacks. But there are also some problems in  $\text{Twig}^2\text{Stack}$ . First, the way of maintaining ancestor/descendant relationships across the hierarchical-stacks is too complex, which results in a large number of random memory accesses and therefore increases the processing time. Second,  $\text{Twig}^2\text{Stack}$  needs to maintain a large number of stacks. In the worst case, it needs to load the whole XML tree into memory.

## 4 A New Algorithm: *TwigList*

We have developed a new algorithm *TwigList* for processing twig-pattern matching queries. The main difference between our *TwigList* algorithm and  $\text{Twig}^2\text{Stack}$  is that we do not need to maintain complicated hierarchical-stacks for nodes in a query tree  $Q$ . Instead of maintaining a hierarchical-stack for a node,  $V_i$ , in the query tree, *TwigList* simply maintains a list,  $L_{V_i}$ . The list is used based on the following remark.

*Property 1.* Consider  $A//B$  against an XML tree  $T$ . If an  $A$ -typed node is an ancestor of a set of  $B$ -typed nodes in XML tree  $T$ : (1) It must be able to specify a minimal interval for the  $A$ -typed node to cover all such  $B$ -typed nodes; (2) It must be the case that there does not exist any  $B$ -typed node in the interval that is not a descendant of the  $A$ -typed node.

---

**Algorithm 1.** *TwigList* ( $Q, \mathcal{T}$ )

---

**Input:** a query tree  $Q$  with  $n$  nodes  $\{V_1, \dots, V_n\}$ , and an XML tree  $\mathcal{T}$ ;  
**Output:** all  $n$ -ary tuples as answers for  $Q$ ;

- 1: let  $X_i$  be a sequence of  $V_i$ -typed XML tree nodes sorted in preorder, for all  $1 \leq i \leq n$ ;
- 2: let  $X$  be the set of all  $X_i$  for  $1 \leq i \leq n$ ;
- 3: obtain a set of lists  $L = \{L_{V_1}, L_{V_2}, \dots, L_{V_n}\}$  by calling *TwigList-Construct* ( $Q, X$ );
- 4:  $R \leftarrow \text{TwigList-Enumerate} (Q, L)$ ;
- 5: **return**  $R$ ;

---

It is important to know that existing algorithms *TwigStack* and *Twig<sup>2</sup>Stack* do not fully make use of this property. Push/pop operations together with stacks cannot effectively maintain this property. We fully and effectively make use of this property. Unlike *TwigStack* and *Twig<sup>2</sup>Stack*, we mainly use lists instead of stacks. Unlike *TwigStack*, we minimize the cost of enumerating results to the minimum (linear time), because the merging procedure of  $n$  joins is avoided. Unlike *Twig<sup>2</sup>Stack*, we do not need to use complex hierarchical-stacks, and maximize the possibility to conduct sequential scans over the lists. When generating outputs for a twig-pattern matching query,  $Q$ , by enumerating the generated lists, we do not need to use any extra memory space, which further saves cost. Our algorithm is optimal in the sense that both time and space complexities of our algorithm are linear w.r.t. the total number of occurrences of twig-pattern matchings and the size of XML tree. As shown in our experimental studies, our external *TwigList* algorithm outperforms *Twig<sup>2</sup>Stack* as well as *TwigStack*.

*TwigList* algorithm is outlined in Algorithm 1 which takes two inputs, a query tree  $Q(V, E)$ , representing a twig-pattern matching query, and an XML tree,  $\mathcal{T}$ . The query tree has  $n$  nodes,  $\{V_1, V_2, \dots, V_n\}$ . *TwigList* constructs lists for all  $V_i$ -typed nodes in  $\mathcal{T}$ , and sorts them following preorder. There are two main steps. First, it calls *TwigList-Construct* to obtain a set of lists that compactly maintain all twig-patterns for answering  $Q$  (line 3). Second, it calls *TwigList-Enumerate* to obtain all  $n$ -ary tuples for  $Q$  (line 4). In the following, we discuss the two main algorithms, *TwigList-Construct* and *TwigList-Enumerate*, in detail. For simplicity, we first concentrate on query trees,  $Q$ , where only // edges appear. Then, we will discuss how to process a query tree with // edges as well as / edges.

#### 4.1 *TwigList-Construct* Algorithm

*TwigList-Construct* is outlined in Algorithm 2. We explain how *TwigList* works using an example of the same twig-pattern matching query  $Q = //A[//C]//B$  (Fig. 1(b)) against XML tree  $\mathcal{T}$  (Fig. 1(a)). In  $Q$ , there are three types  $A$ ,  $B$ , and  $C$ .  $A$  is the root node, and  $B$  and  $C$  are leaf nodes. Accordingly, as input,  $X$  consists of three sequences for the three types,  $X_A = \langle a_1, a_2, a_3 \rangle$ ,  $X_B = \langle b_1, b_2, b_3, b_4 \rangle$ , and  $X_C = \langle c_1, c_2, c_3 \rangle$ . *TwigList-Construct* will generate three lists,  $L_A$ ,  $L_B$  and  $L_C$ , to determine all possible  $n$ -ary tuples for answering  $Q$ . Here, for this  $Q$ , every XML tree node,  $a_i$ , in  $L_A$  will maintain two pairs of pointers to specify the intervals for its  $B$ -typed descendants,  $(start_B, end_B)$ , and its  $C$ -typed descendants,  $(start_C, end_C)$ .

**Algorithm 2.** *TwigList-Construct* ( $Q, X$ )

**Input:** a query tree  $Q$  with  $n$  nodes  $\{V_1, \dots, V_n\}$ , and a set of sequences  $X$ , a sequence in  $X$ ,  $X_i$ , maintains a list of  $V_i$ -typed XML nodes;

**Output:** all  $L_{V_i}$  for each  $1 \leq i \leq n$ .

```

1: initialize stack S as empty;
2: initialize all list L_{V_i} as empty for all $V_i \in V(Q)$ (The length of L_{V_i} is initialized as 0);
3: while not all $X_q = \emptyset$, for $1 \leq q \leq n$, do
4: let V_q be the node such that its top element is the first following the preorder traversal
 among all top elements in all X_i ;
5: let v be the top element in X_q ;
6: remove v from X_q ;
7: $toList(S, Q, reg(v))$;
8: for each child of V_q in query tree Q , V_p , do
9: $v.start_{V_p} \leftarrow length(L_{V_p}) + 1$;
10: push(S, v);
11: $toList(S, Q, (\infty, \infty))$;
```

**Procedure**  $toList(S, Q, r)$

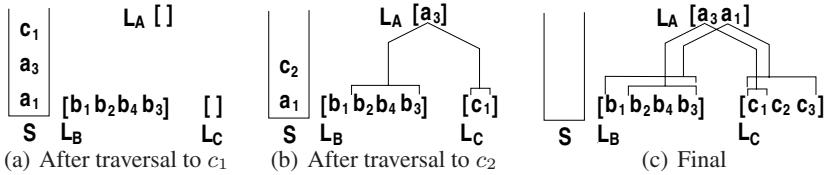
```

12: while $S \neq \emptyset \wedge r \not\subseteq reg(top(S))$ do
13: $v_j \leftarrow pop(S)$;
14: let v_j 's type be V_j ;
15: for each child of V_j in query tree Q , V_k , do
16: $v_j.end_{V_k} \leftarrow length(L_{V_k})$;
17: append v_j into list L_{V_j} if $v_j.start_{V_k} \leq v_j.end_{V_k}$ for every V_j 's child, V_k ;
```

---

Initially, it initializes a working stack  $S$  to be empty (line 1), and create empty lists,  $L_A$ ,  $L_B$ , and  $L_C$  (line 2). Below, we use  $length(L_X)$  to indicate the length of the list  $L_X$ . All the lengths of the lists are zero. In line 3-10, it repeats until all sequences,  $X_A$ ,  $X_B$ , and  $X_C$ , become empty. In every iteration, *TwigList-Construct* selects a node from the sequences that is the first following preorder (line 4-6). For this example, *TwigList-Construct* access  $a_1, a_2, b_1, a_3, b_2, b_3, b_4, c_1, c_2$ , and  $c_3$  in order, and will push them into  $S$ .

Suppose  $a_1, a_2$  and  $b_1$  are pushed into  $S$  already.  $a_1$  and  $a_2$ 's  $start_B$  and  $start_C$  pointers will point to the ends of  $L_B$  and  $L_C$  ( $length(L_B) + 1$  and  $length(L_C) + 1$ ), respectively. Their  $end_B$  and  $end_C$  will be updated later, because they are unknown now. When *TwigList-Construct* is about to push  $a_3$  into  $S$ , it calls  $toList$  (line 7) with its region-code  $reg(a_3)$ . The body of  $toList$  is from line 12 to line 17.  $toList$  finds that  $b_1$  as the top element in  $S$  is not an ancestor of  $a_3$  and therefore pops-up  $b_1$  from  $S$  and appends  $b_1$  to  $L_B$ . Here,  $a_3$ 's  $start_B$  will point to  $length(L_B) + 1$ , because  $b_1$  is not a descendant of  $a_3$  and  $a_3$ 's  $B$ -typed descendants will come after it, if any.  $a_3$ 's  $start_C$  will point to  $L_C$  ( $length(L_C) + 1$ ) which is still empty. Then,  $toList$  also finds that  $a_2$  as the current top element in  $S$  is not an ancestor of  $a_3$ .  $toList$  does not append it into  $L_A$  because it does not have any  $C$ -typed descendants now, and will not have any later (line 17). When *TwigList-Construct* is about to push  $b_3$  into  $S$  after pushing  $b_2$  into  $S$ ,  $toList$  finds that  $b_2$  is not  $b_3$ 's ancestor, it pops up  $b_2$  from  $S$  and appends  $b_2$  to  $L_B$ .



**Fig. 3.** *TwigList-Construct* for the query tree  $Q$  (Fig. 1(b)) against XML tree  $\mathcal{T}$  (Fig. 1(a))

Fig. 3(a) shows the stack  $S$  and the lists,  $L_A$ ,  $L_B$ , and  $L_C$ , after  $c_1$  is pushed into  $S$ . Fig. 3(b) shows  $S$  and the lists after  $c_2$  is pushed into  $S$ . When  $c_2$  is pushed into  $S$ ,  $toList$  enforces  $c_1$  and  $a_3$  to be popped up, and append to the corresponding lists. This is the timing for  $a_3$  to fill in its  $end_B$  and  $end_C$  positions ( $length(L_B)$  and  $length(L_C)$ ), respectively. Fig. 3(c) shows the stack  $S$  and the lists after all XML tree nodes are pushed/popped-up into/from  $S$ . As can be seen from Fig. 3(c), all twig-patterns are maintained by the lists.

In line 11, *TwigList-Construct* uses  $(\infty, \infty)$  as the largest region code to enforce all in stack  $S$  to be appended into a list if possible.

**Time/Space Complexity:** Given a twig-pattern matching query,  $Q$ , and an XML tree  $\mathcal{T}$ . Suppose the corresponding query tree,  $Q$ , has  $n$  nodes,  $V_1, V_2, \dots, V_n$ . The time/space complexity of *TwigList-Construct* (Algorithm 2) are both  $O(d \cdot |X|)$  in the worst case, where  $|X|$  is the total number of nodes,  $v_i$ , in XML tree that is  $V_i$ -typed  $1 \leq i \leq n$ , and  $d$  is the max degree of a node in the query tree  $Q$ . Note: every XML tree node that is  $V_i$ -typed will be pushed/popped-up into/from the stack  $S$  only once. It needs at most  $d$  times to calculate its intervals. Therefore, *TwigList-Construct* is linear w.r.t.  $|X|$ .

## 4.2 TwigList-Enumerate Algorithm

*TwigList-Enumerate* is outlined in Algorithm 3. It takes two input parameters, the  $n$ -node query tree  $Q$  and a set of lists,  $L$ , which consists of the lists obtained in *TwigList-Construct*. *TwigList-Enumerate* simply inserts all  $n$ -ary tuples as answers into a relation  $R$  to be returned.

Continue the above example, the three lists,  $L_A$ ,  $L_B$ , and  $L_C$ , are shown in Fig. 3(c). Initially,  $start = [a_3, b_2, c_1]$ , and  $end = [a_1, b_3, c_1]$  (line 2-4). Here, pair  $(a_3, a_1)$  specifies the interval for all  $A$ -typed XML tree nodes.  $(b_2, b_3)$  specifies the interval where  $a_3$ 's  $B$ -typed descendants exist, and  $(c_1, c_1)$  specifies the interval where  $a_3$ 's  $C$ -typed descendants exist. Line 5,  $move = [a_3, b_2, c_1]$  records the current positions for outputting results. Then, it calls *moreMatch* to generate all  $n$ -ary tuples. In *moreMatch*, it first inserts the  $n$ -ary tuple pointed by the  $n$ -element *move* array. The termination condition is specified in line 9 when there is no tuple to be generated. *TwigList-Enumerate* will result  $(a_3, b_2, c_1)$ ,  $(a_3, b_4, c_1)$ , and  $(a_3, b_3, c_1)$ , followed by  $(a_1, b_1, c_1), \dots$

**Time/Space Complexity:** Given a twig-pattern matching query,  $Q(V, E)$ , as a query tree, and an XML tree  $\mathcal{T}$ . Suppose lists  $L_{V_1}, L_{V_2}, \dots, L_{V_n}$  have been constructed, the time complexity of *TwigList-Enumerate* algorithm is  $O(n \cdot |R|)$ , where  $|R|$  is the total number of twig-pattern matchings, and  $n$  is the number of nodes in query tree,  $Q$ .

**Algorithm 3.** *TwigList-Enumerate* ( $Q, L$ )

**Input:** a query tree,  $Q$ , with  $n$  nodes  $\{V_1, \dots, V_n\}$ ; a set of lists,  $L$ , consisting of all  $L_{V_i}$ , for  $1 \leq i \leq n$ ;

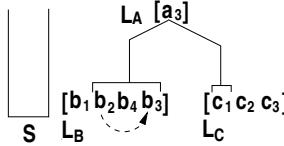
**Output:** all  $n$ -ary tuples as answers for  $Q$ ;

- 1: let  $R \leftarrow \emptyset$ ;
  - 2: let  $start[1..n]$ ,  $end[1..n]$  be  $n$ -element arrays for maintaining the regions for the  $n$  lists;
  - 3: let  $V_1$  be the root node in the query tree  $Q$ ;  $start[1]$  and  $end[1]$  point to the begin and end positions of  $L_{V_1}$ ;
  - 4: initialize the other  $start[i]$  and  $end[i]$  for  $V_i$  as the interval specified by the first element in its parent list;
  - 5: let  $move[1..n]$  be a  $n$ -element array where  $move[i] \leftarrow start[i]$ ;
  - 6: **while**  $moreMatch(R, start, end, move) \neq \text{false}$  do;
  - 7: **return**  $R$ ;
- Function** *moreMatch*( $R, start, end, move$ )
- 8: insert ( $move[1], \dots, move[n]$ ) as a  $n$ -ary tuple into  $R$ ;
  - 9: **if**  $\forall i: move[i] = end[i]$  **then return** false;
  - 10: select  $V_i$  such that  $move[i] < end[i]$ , but all its descendants,  $V_j$ , in the query tree  $Q$ ,  $move[j] = end[j]$ ;
  - 11:  $move[i] \leftarrow move[i] + 1$ ;
  - 12: let  $v_i$  be the  $V_i$ -typed element pointed by  $move[i]$ ;
  - 13: **for** all  $V_i$ 's descendants,  $V_j$ , in query tree  $Q$  **do**
  - 14:     reset all their  $start[j]$ ,  $end[j]$ , and  $move[j]$  according to the interval specified by its parent (rooted at  $v_i$ );
  - 15: **return** true;

Because in each run of function *moreMatch*, we will get one more matching (line 8), and the operations below in *moreMatch* require time  $O(n)$ . The space complexity of *TwigList-Enumerate* is the same for *TwigList-Construct*, because it does not consume any more memory space, other than three arrays  $start[1..n]$ ,  $end[1..n]$ , and  $move[1..n]$ . Hence, the time complexity for *TwigList* is the sum of that for *TwigList-Construct* and *TwigList-Enumerate*,  $O(d \cdot |X| + n \cdot |R|)$ . This algorithm is optimal because it is linear w.r.t.  $|R|$  and  $|X|$ . Note  $O(n \cdot |R|)$  is lower bound to output all twig-pattern matchings of a  $n$ -node query tree explicitly.

### 4.3 Discussions

**Handling /-Edges in Query Trees:** If there are /-edges in a query tree,  $Q$ , it needs to have additional information to maintain the sibling information for efficiently processing twig-pattern matching queries. Consider  $Q' = //A//C/B$  against the XML tree  $T$  (Fig. 1(a)). Only  $(a_3, b_2, c_1)$  and  $(a_3, b_3, c_1)$  are the answers of  $Q'$ . We can simply extend *TwigList-Construct* to construct lists when there are /-edges in a given query tree. For  $Q' = //A//C/B$ , the lists constructed are shown in Fig. 4. As shown in Fig. 4, there is no need to append  $a_1$  into list  $L_A$ , because  $a_1$  does not have a  $B$ -typed child when it is about to append. When  $b_3$  is about to be appended into  $L_B$ , *TwigList-Construct* knows that  $b_2$  is a sibling which shares the same  $A$ -typed parent of  $b_3$ , a



**Fig. 4.** *TwigList-Construct* for  $Q' = //A//C/B$  against XML tree  $\mathcal{T}$  (Fig. 1(a))

sibling link can be added from  $b_2$  to  $b_3$ . Note: some  $b_i$  and  $c_j$  are not in the interval of  $a_3$  as shown in Fig. 4, it is because that it is unknown whether it is in the interval of its parent when it is appended into the corresponding list. With the sibling pointers, *TwigList-Enumerate* can quickly enumerate all results.

**External Algorithm:** When the set of lists  $L$  is too large to fit into memory, *TwigList* can be simply extended to work as an external algorithm by maintaining all lists on disk. It is because the access patterns against the lists usually focus on intervals and are not random. We implemented an external *TwigList* algorithm with  $n$  4KB-pages for a query tree with  $n$  nodes. The external *TwigList* algorithm outperforms *Twig<sup>2</sup>Stack*.

## 5 Performance Study

We have implemented three algorithms for processing twig-pattern matching queries: *Twig<sup>2</sup>Stack* [9], our *TwigList*, and our external version of *TwigList* (*E-TwigList*) using C++. We choose *Twig<sup>2</sup>Stack* as the basis to compare, because *Twig<sup>2</sup>Stack* is the most up-to-date algorithm which outperforms *TwigStack* [3] and *TJFast* [6]. *TJFast* is a fast algorithm for processing twig-pattern matching queries with both  $//$ -edges and  $/$ -edges.

**Three Datasets:** We used both benchmark dataset, XMark, and two real datasets, DBLP and TreeBank. For XMark, we set the scaling factor to be 5.0 and generated a 582MB XMark dataset with 77 different labels and a maximum depth of 12. For real datasets, we use a 337MB DBLP dataset which has 41 different labels and a maximum depth of 6, and the 84MB TreeBank dataset which has 250 different labels and a maximum depth of 36. The DBLP dataset is wide but shallow, whereas the TreeBank dataset is deep and recursive.

All experiments were performed on a 2.8G HZ Pentium (R)4 processor PC with 1GB RAM running on Windows XP system. We mainly report processing time for construction and enumeration used in *Twig<sup>2</sup>Stack*, *TwigList*, and *E-TwigList*, since the other time as loading and storing final results are the same. The buffer size used for *E-TwigList* is a 4KB-page for every node in a query tree.

**Twig-pattern matching queries:** We conducted extensive testing, and report the results for 15 twig-pattern matching queries (query trees) as shown in Table 1. For each of the three datasets, we report five query trees, which have different combinations of  $/$ -edges and  $//$ -edges and different selectivities. In each group of 5 query trees, the first 2 are selected from the queries used in [9], and the second 2 are constructed by adding some branches into the first 2. The last is a rather complex query tree.

**Table 1.** A list of query trees used in the experiments

| Name | Dataset  | QueryTrees                                                       | ResultSize  |
|------|----------|------------------------------------------------------------------|-------------|
| XQ1  | XMark    | //item[location]/description//keyword                            | 136, 282    |
| XQ2  | XMark    | //people//person[./address zipcode]/profile/education            | 15, 859     |
| XQ3  | XMark    | //item[location][./mailbox/mail//emph]/description//keyword      | 86, 533     |
| XQ4  | XMark    | //people//person[./address zipcode][id]/profile[./age]/education | 7, 997      |
| XQ5  | XMark    | //open_auction//annotation[./person]/parlist//bidder//increase   | 141, 851    |
| DQ1  | DBLP     | //dblp/inproceedings[title]/author                               | 1, 205, 196 |
| DQ2  | DBLP     | //dblp/article[author][./title]/year                             | 625, 991    |
| DQ3  | DBLP     | //dblp/inproceedings[./cite/label][title]/author                 | 132, 902    |
| DQ4  | DBLP     | //dblp/article[author][./title][./url][./ee]/year                | 384, 474    |
| DQ5  | DBLP     | //article[./mdate][./volume][./cite/label]/journal               | 13, 785     |
| TQ1  | TreeBank | //S/VP//PP[./NP/VBN]/IN                                          | 1, 183      |
| TQ2  | TreeBank | //S/VP/PP[IN]/NP/VBN                                             | 152         |
| TQ3  | TreeBank | //S/VP//PP[./NN][./NP[./CD]/VBN]/IN                              | 381         |
| TQ4  | TreeBank | //S[./VP][./NP]/VP/PP[IN]/NP/VBN                                 | 1, 185      |
| TQ5  | TreeBank | //EMPTY[./VP/PP//NNP][./S[./PP/JJ]/VBN]/PP/NP//NONE..            | 94, 535     |

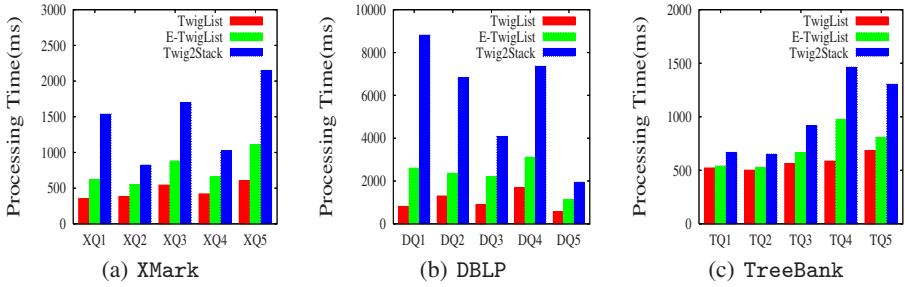
**Fig. 5.** Processing Time (ms)

Fig. 5 depicts the processing time of query trees listed in Table 1 for the datasets, XMark (Fig. 5(a)), DBLP (Fig. 5(b)), and TreeBank (Fig. 5(c)). *TwigList* and even *E-TwigList* outperform *Twig<sup>2</sup>Stack* in all tests. *TwigList* (*E-TwigList*) outperforms *Twig<sup>2</sup>Stack*, mainly due to the linear structure (lists) used to organize the elements instead of complex hierarchical-stacks used in *Twig<sup>2</sup>Stack*. Also, when enumerating results, *Twig<sup>2</sup>Stack* uses a join approach which produces a lot of intermediate results, whereas our *TwigList* (*E-TwigList*) does not generate any intermediate results.

For XMark (Fig. 5(a)), on average, *TwigList* is 3-4 times and *E-TwigList* is 2-3 times faster than *Twig<sup>2</sup>Stack*. For DBLP (Fig. 5(b)), on average, *TwigList* is 4-8 times and *E-TwigList* is 2-4 times faster than *Twig<sup>2</sup>Stack*. For TreeBank (Fig. 5(c)), *TwigList* and *E-TwigList* outperform *Twig<sup>2</sup>Stack*, in particular when the query tree becomes complex, for example, TQ5. Our algorithms based on linear structures (lists) replace a large number of random accesses with sequential accesses in both memory and disk.

**E-TwigList Test:** We further test *E-TwigList* by choosing three queries, XQ3, DQ3 and TQ3, as representations of the queries over XMark, DBLP and TreeBank datasets. Their structures are moderately complex, and they produce a moderate number of matchings. XQ3 has 7 nodes with a tree of depth 4 and max node degree 3, DQ3 has 6 nodes with a tree of depth 4 and max degree 3, and TQ3 has 8 nodes with a tree of depth 5 and max degree 3. The total number of I/Os include the I/O cost in loading, construction and

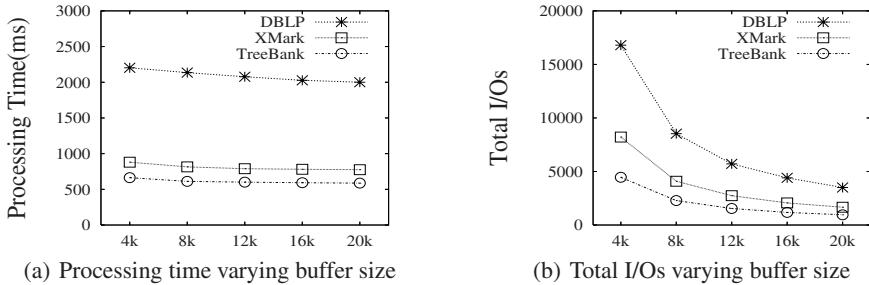


Fig. 6. Total Processing time and I/Os varying buffer size

enumeration. We vary the buffer size from 4KB to 20KB. As shown in Fig. 6(b), we can see that the total number of I/Os decreases when the buffer size increases. We can also see from Fig. 6(a) that the processing time decreases when the buffer size increases, but the effect of buffer sizes on processing times is not obvious. It concludes that only a small buffer is needed for a node in a query tree.

## 6 Conclusion and Future Work

In this paper, we propose a new *TwigList* algorithm to compactly maintain the twig-patterns using simple lists. The algorithm can be easily extended as an external algorithm (*E-TwigList*). The time and space complexity of the algorithm are linear with respect to the total number of occurrences of twig-patterns and the size of XML tree. Our algorithm significantly outperforms *Twig<sup>2</sup>Stack* algorithm.

As the future work, we are planning to study the pattern matching over directed acyclic graphs using the similar method to get a better performance.

**Acknowledgment.** This work was supported by a grant of RGC, Hong Kong SAR, China (No. 418206).

## References

1. Zhang, C., Naughton, J.F., DeWitt, D.J., Luo, Q., Lohman, G.M.: On supporting containment queries in relational database management systems. In: SIGMOD. (2001)
2. Al-Khalifa, S., Jagadish, H.V., Patel, J.M., Wu, Y., Koudas, N., Srivastava, D.: Structural joins: A primitive for efficient XML query pattern matching. In: ICDE. (2002)
3. Bruno, N., Koudas, N., Srivastava, D.: Holistic twig joins: optimal XML pattern matching. In: SIGMOD. (2002)
4. Jiang, H., Lu, H., Wang, W., Ooi, B.C.: Xr-tree: Indexing XML data for efficient structural joins. In: ICDE. (2003)
5. Lu, J., Chen, T., Ling, T.W.: Efficient processing of XML twig patterns with parent child edges: a look-ahead approach. In: CIKM. (2004)

6. Lu, J., Ling, T.W., Chan, C.Y., Chen, T.: From region encoding to extended dewey: On efficient processing of XML twig pattern matching. In: VLDB. (2005)
7. Choi, B., Mahoui, M., Wood, D.: On the optimality of holistic algorithms for twig queries. In: DEXA. (2003)
8. Aghili, S., Li, H.G., Agrawal, D., Abbadi, A.E.: Twix: Twig structure and content matching of selective queries using binary labeling. In: INFOSCALE. (2006)
9. Chen, S., Li, H.G., Tatenuma, J., Hsiung, W.P., Agrawal, D., Candan, K.S.: Twig2stack: Bottom-up processing of generalized-tree-pattern queries over XML documents. In: VLDB. (2006)

# CircularTrip: An Effective Algorithm for Continuous $k$ NN Queries

Muhammad Aamir Cheema, Yidong Yuan, and Xuemin Lin

The University of New South Wales, Australia  
`{macheema, yyidong, lxue}@cse.unsw.edu.au`

**Abstract.** Continuously monitoring  $k$ NN queries in a highly dynamic environment has become a necessity to many recent location-based applications. In this paper, we study the problem of continuous  $k$ NN query on the dataset with an in-memory grid index. We first present a novel data access method – CircularTrip. Then, an efficient CircularTrip-based continuous  $k$ NN algorithm is developed. Compared with the existing algorithms, our algorithm is both space and time efficient.

## 1 Introduction

Continuously monitoring  $k$  nearest neighbors over moving data objects has become a necessity to many recent location-based applications. This is mainly due to the increasing availability of wireless networks and inexpensive mobile devices. Consequently, a number of techniques [1,2,3,4,5,6,7,8,9] have been developed to process continuous  $k$ NN queries.

Different from a conventional  $k$ NN query, continuous  $k$ NN queries are issued once and run continuously to generate results in real-time along with the updates of the underlying datasets. Therefore, it is crucial to develop in-memory techniques to continuously process  $k$ NN queries due to frequent location updates of data points and query points. In many applications [6,7,9], it is also crucial to support the processing of a number of continuous  $k$ NN queries simultaneously; consequently, scalability is a key issue.

To address the scalability, in this paper we focus on two issues: (1) minimization of computation costs; and (2) minimization of the memory requirements. We study continuous  $k$ NN queries against the data points that move around in an arbitrary way. To effectively monitor  $k$ NN queries, we develop a novel data access method – CircularTrip. Compared with the most advanced algorithm, CPM [9], our CircularTrip-based continuous  $k$ NN algorithm has the following advantages. (1) time efficient: although both algorithms access the minimum number of cells for initial computation, less cells are accessed during continuous monitoring in our algorithm. (2) space efficient: our algorithm does not employ any book-keeping information used in CPM (i.e., visit list and search heap for each query). Our experimental study demonstrates that CircularTrip-based continuous  $k$ NN algorithm is 2 to 4 times faster than CPM, while its memory usage is only 50% to 85% of CPM.

Our contributions in this paper can be summarized as follows: (1) We develop a novel data access method – CircularTrip which returns the cells intersecting a given circle with the minimum number of cells examined; (2) Based on CircularTrip, a time- and space- efficient continuous  $k$ NN algorithm is developed.

The rest of the paper is organized as follows: Section 2 gives the problem definition and presents the related work. We present our continuous  $k$ NN algorithm in Section 3. Experimental study and remarks are reported in Section 4.

## 2 Background Information

Suppose that  $P$  is a set of 2D<sup>1</sup> moving data points and data points change their locations frequently in an unpredictable fashion. Each data point  $p \in P$  is represented by  $(p.x, p.y)$ . At each time stamp, the move of a data point  $p$  from  $p_{pre}$  to  $p_{cur}$  is recorded as a location update  $\langle p.id, p_{pre}, p_{cur} \rangle$  and the moves of query points are recorded similarly. The problem of continuous  $k$ NN query is formally defined below.

**Continuous  $k$ NN Query.** *Given a set of moving data points, a moving query point  $q$ , and an integer  $k$ , the continuous  $k$ NN query is to find  $k$  closest data points to  $q$  at each time stamp.*

**Grid Index.** In this paper, we assume that the dataset is indexed by an in-memory grid index which evenly partitions the space into *cells*. The extent of each cell on each dimension is  $\delta$ . Cell  $c[i, j]$  indicates the cell at column  $i$  and row  $j$  and the lower-left corner cell is  $c[0, 0]$ . Clearly, point  $p$  falls into the cell  $c[\lfloor p.x/\delta \rfloor, \lfloor p.y/\delta \rfloor]$ .

In the grid index, each cell is associated with an *object list* and an *influence list*. Object list contains all the data points in this cell. Influence list of cell  $c$  maintains the references of all the queries  $q$  such that  $mindist(c, q) \leq q.dist_k$  where  $q.dist_k$  is the distance of  $k^{th}$  nearest neighbor from  $q$ . Specially,  $mindist(c^q, q) = 0$  where  $c^q$  is the cell containing  $q$ . Note that both object list and influence list are implemented as hash tables so that lookup, insertion, update, and deletion of entries take constant time.

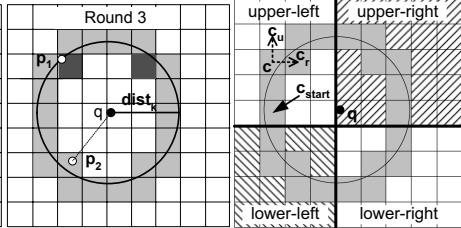
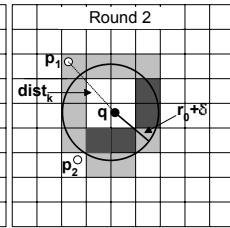
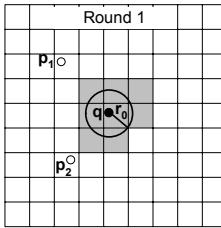
*Accessing* and *encountering* are two basic operations on cells. Specifically, accessing a cell is to evaluate all data points in this cells against queries and encountering a cell only computes its minimum distance to queries. Clearly, cost of encountering a cell is neglected when compared with accessing a cell.

SEA-CNN [6], YPK-CNN [7], and CPM [9] are the most related existing work to the study in this paper. Due to space limitation, we omit the details of these techniques here. Interested readers can find them in [6, 7, 9], respectively.

## 3 Continuous $k$ NN Algorithm

Our CircularTrip-based continuous  $k$ NN algorithm consists of two phases. In phase 1, the initial results of each new continuous  $k$ NN query is computed. Then,

<sup>1</sup> In this paper, we focus on 2D space only. But the proposed techniques can be applied to higher dimensional space immediately.



**Fig. 1.** An NN Query

**Fig. 2.** CircularTrip

the results are incrementally updated by continuous monitoring module at each time stamp upon the moves of query points and data points (i.e., phase 2). Both phases take advantages of CircularTrip algorithm. Section 3.1 and Section 3.2 present phase 1 and phase 2, respectively.

### 3.1 Initial $k$ NN Computation

The basic idea of  $k$ NN computation algorithm is to access the cells around query point  $q$  round by round. A *round*  $C_i$  contains all the cells that intersect the circle of radius  $r_i = r_0 + i\delta$  centered at  $q$ . Formally,  $C_i = \{\forall c \mid \text{mindist}(c, q) < r_i \leq \text{maxdist}(c, q)\}$ .  $r_0$  is the first circle's radius. Obviously,  $r_0$  is at most  $\text{maxdist}(c^q, q)$ ; otherwise cell  $c^q$  will not be accessed by the algorithm. Examples of round are shown as the shaded cells in Fig. 1. In each round, the algorithm accesses the cells in ascending order of their  $\text{mindist}(c, q)$ . The algorithm terminates when the next cell to be accessed has  $\text{mindist}(c, q) \geq q.dist_k$ . The following theorem proves the correctness and optimality of this algorithm.

**Lemma 1.** *In a grid consisting of cells with size  $\delta \times \delta$ , given a cell  $c$  and a query point  $q$  where  $c$  does not contain  $q$ ,  $\delta \leq \text{maxdist}(c, q) - \text{mindist}(c, q) \leq \sqrt{2}\delta$ .  $\square$*

**Theorem 1.** *Given a query  $q$ , in our initial  $k$ NN algorithm, the minimal set of cells are all accessed and only these cells are accessed.  $\square$*

According to Lemma 1, a cell is intersected by at most two consecutive circles (e.g., the dark shaded cells in Fig. 1). Although these cells are encountered twice during  $k$ NN computation (i.e., these cells appear in two rounds), they are accessed once only. This is because for a query  $q$  (1) our  $k$ NN algorithm only accesses the cells where  $q$  is not in their influence lists; and (2)  $q$  will be inserted into its influence list after a cell is processed. In fact, Theorem 2 proves the upper bound of the total number of times the cells are encountered in our algorithm.

**Theorem 2.** *In  $k$ NN algorithm, the total number of times the cells are encountered is at most 1.27 times of the number cells in the minimum set of cells.  $\square$*

The detailed  $k$ NN computation algorithm is shown in Algorithm 1. We use the following example to present its details.

**Algorithm 1.** ComputeNN( $G, q, k$ )

---

**Input:**  $G$ : the grid index;  $q$ : query point;  $k$ : an integer;  
**Output:** the  $k$ NN of  $q$ ;

- 1:  $q.dist_k := \infty$ ;  $q.kNN := \emptyset$ ;  $H := \emptyset$ ;  $r := r_0 := maxdist(c^q, q)$ ;
- 2: insert the cells returned by **CircularTrip**( $G, q, r$ ) into  $H$ ;
- 3: **while**  $H \neq \emptyset$  and  $mindist(e^H, q) < q.dist_k$  **do**
- 4:   insert  $q$  into the influence list of  $e^H$ ;
- 5:    $\forall p \in e^H$ , compute  $dist(p, q)$  and update  $q.dist_k$  and  $q.kNN$ ;
- 6:   remove  $e^H$  from  $H$ ;
- 7:   **if**  $H = \emptyset$  and  $r < q.dist_k$  **then**
- 8:      $r := \min\{r + \delta, q.dist_k\}$ ;
- 9:     cells  $C := \text{CircularTrip}(G, q, r)$ ;
- 10:     $\forall c \in C$ , insert  $c$  into  $H$  if  $q \notin$  the influence list of  $c$ ;
- 11: **return**  $q.kNN$ ;

---

*Example 1.* Fig. 11 illustrates a concrete example of an NN query. As no data point is found in the first round, the algorithm continues to process the cells in the next round with radius ( $r_0 + \delta$ ). In this round,  $p_1$  is found and  $q.dist_k$  is updated to be  $dist(p_1, q)$ . Then, a third round with radius  $q.dist_k$  (as  $dist(p_1, q) < r_0 + 2\delta$ ) is processed because the previous radius is smaller than  $q.dist_k$ . In round 3,  $q.kNN$  and  $q.dist_k$  are updated after  $p_2$  is found. Computation stops when  $q.dist_k$  ( $= dist(p_2, q)$ ) is less than  $mindist(e^H, q)$  of the top entry  $e^H$ .

**CircularTrip Algorithm.** To collect a round of cells, CircularTrip starts from one cell intersected by the given circle and checks the cells along the circle. Without loss of generality, consider cell  $c$  intersected by the circle which locates in the upper-left quadrant as shown in Fig. 12. The key fact is that the next cell intersected by the circle (i.e., the cell in which the arc is connected to one in  $c$ ) is the adjacent cell either above  $c$  (i.e.,  $c_u$ ) or right to  $c$  (i.e.,  $c_r$ ). This is because the outgoing circle crosses either the upper boundary or the right boundary of  $c$ . These two adjacent cells,  $c_u$  and  $c_r$ , are called *candidate adjacent cells* of  $c$ . Clearly, to collect the next cell intersected by the circle, CircularTrip only needs to examine one of the candidate adjacent cells (i.e., check its  $mindist(c, q)$  with the given radius  $r$ ). As a result, the total cost of CircularTrip to collect a round  $C$  of cells is to compute  $mindist(c, q)$  of  $|C|$  cells, where  $|C|$  is the number of cells in round  $C$ .

Algorithm 2 presents the implementation of CircularTrip algorithm. It always starts from the left most cell of the round  $c_{start}$  (as shown in Fig. 12) and examines the cells clockwise along the given circle until  $c_{start}$  is encountered again. When the quadrant of the current cell being examined is changed, the directions to find its candidate adjacent cells are updated accordingly (i.e., lines 9 – 10).

### 3.2 Continuous Monitoring

Same as in CPM, when the query moves, we simply re-issue the query on the new location. So, continuous monitoring only concerns update of data points.

**Algorithm 2.** CircularTrip( $G, q, r$ )

---

**Input:**  $G$ : the grid index;  $q$ : query point;  $r$ : the radius;  
**Output:** all the cells which intersect the circle with center  $q$  and radius  $r$ ;

- 1:  $C := \emptyset$ ;  $c := c_{start} := c[i, j]$  ( $i := \lfloor (q.x - r)/\delta \rfloor$ ,  $j := \lfloor q.y/\delta \rfloor$ );
- 2:  $D_{cur} := Up$ ; /\* clockwise fashion:  $Up \rightarrow Right \rightarrow Down \rightarrow Left \rightarrow Up$  \*/
- 3: **repeat**
- 4:   insert  $c$  into  $C$ ;
- 5:    $c' :=$  the adjacent cell to  $c$  in  $D_{cur}$  direction;
- 6:   **if**  $c'$  does not intersect the circle **then**
- 7:      $c' :=$  the adjacent cell to  $c$  in the next direction of  $D_{cur}$ ;
- 8:      $c := c'$ ;
- 9:   **if**  $(c.i = c^q.i$  and  $c.j = \lfloor (q.y \pm r)/\delta \rfloor)$  or  $(c.i = \lfloor (q.x \pm r)/\delta \rfloor$  and  $c.j = c^q.j)$  **then**
- 10:      $D_{cur} :=$  the next direction of  $D_{cur}$ ;
- 11: **until**  $c = c_{start}$
- 12: **return**  $C$ ;

---

Regarding a query  $q$ , the update of data point  $p$ ,  $\langle p.id, p_{pre}, p_{cur} \rangle$ , can be classified into 3 cases:

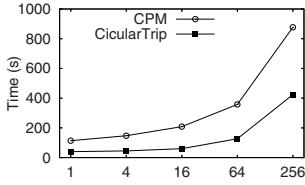
- *internal update*:  $p_{cur} \in q.kNN$  and  $p_{pre} \in q.kNN$ ; clearly, only the order of  $q.kNN$  is affected so we update the order of data points in  $q.kNN$  accordingly.
- *incoming update*:  $p_{cur} \in q.kNN$  and  $p_{pre} \notin q.kNN$ ;  $p$  is inserted in  $q.kNN$ .
- *outgoing update*:  $p_{cur} \notin q.kNN$  and  $p_{pre} \in q.kNN$ ;  $p$  is deleted from  $q.kNN$ .

It is immediately verified that only the queries recorded in the influence lists of cell  $c^{p_{pre}}$  or cell  $c^{p_{cur}}$  may be affected by the update  $\langle p.id, p_{pre}, p_{cur} \rangle$ , where  $c^{p_{pre}}$  ( $c^{p_{cur}}$ ) is the cell containing  $p_{pre}$  ( $p_{cur}$ ). Therefore, after receiving an update  $\langle p.id, p_{pre}, p_{cur} \rangle$ , continuous monitoring module checks these queries  $q$  only. If  $dist(p_{cur}, q) \leq q.dist_k$ , it is treated as an incoming update (if  $p_{pre} \notin q.kNN$ ) or an internal update (if  $p_{pre} \in q.kNN$ ). On the other hand, If  $dist(p_{pre}, q) \leq q.dist_k$  and  $dist(p_{cur}, q) > q.dist_k$ , it is handled as an outgoing update.

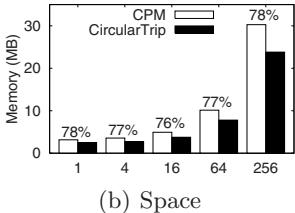
After all the updates of data points are handled as described above, we update the results of affected queries. For each query  $q$ , if  $|q.kNN| \geq k$ , we keep the  $k$  closest points and delete all other. For any query  $q$  where  $|q.kNN| < k$ , we update its result in a similar way to Algorithm 1. Note that here the starting radius  $r_0$  is set as  $q.dist_k$ . The intuition is the fact that any update within this distance has already been handled.

## 4 Experimental Study and Remarks

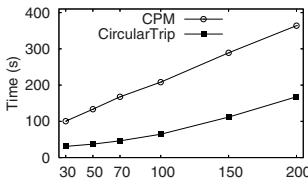
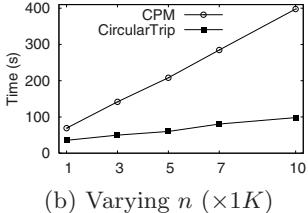
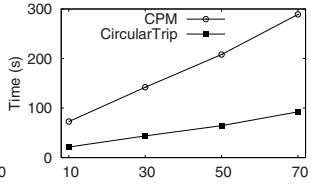
In accordance with the experimental study of previous work [6,9], we use the same spatio-temporal data generator [10]. Data points with slow speed move 1/250 of the extent of space per time stamp. Medium and fast speed are 5 and 25 times faster than slow speed, respectively. Continuous  $k$ NN queries are



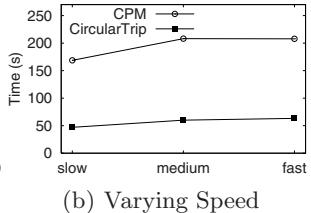
(a) Time



(b) Space

(a) Varying  $N$  ( $\times 1K$ )(b) Varying  $n$  ( $\times 1K$ )

(a) Varying Agility (%)



(b) Varying Speed

**Fig. 3.** Effect of  $k$ **Fig. 4.** Effect of  $N$  and  $n$ **Fig. 5.** Data Movement

generated in the similar way. All queries are evaluated at each time stamp and the length of evaluation is 100 time stamps. The grid index has  $256 \times 256$  cells.

We evaluate our CircularTrip-based continuous  $k$ NN technique against various parameters: number of NNs ( $k$ ), number of data points ( $N$ ), number of queries ( $n$ ), and data point agility and moving speed. In our experiments, their default values are 16,  $100K$ ,  $5K$ , 50%, and *medium*, respectively. The experimental results are reported in Fig. 3, 4, and 5.

In this paper, we develop an efficient CircularTrip-based continuous  $k$ NN algorithm. Compared with the existing algorithm, our technique accesses the minimum set of cells for initial computation and significantly reduces the continuous monitoring cost, while less memory space is required.

## References

1. Song, Z., Roussopoulos, N.: K-nearest neighbor search for moving query point. In: SSTD. (2001) 79–96
2. Tao, Y., Papadias, D.: Time-parameterized queries in spatio-temporal databases. In: SIGMOD Conference. (2002) 334–345
3. Tao, Y., Papadias, D., Shen, Q.: Continuous nearest neighbor search. In: VLDB. (2002) 287–298
4. Zhang, J., Zhu, M., Papadias, D., Tao, Y., Lee, D.L.: Location-based spatial queries. In: SIGMOD. (2003) 443–454
5. Iwerks, G.S., Samet, H., Smith, K.P.: Continuous  $k$ -nearest neighbor queries for continuously moving points with updates. In: VLDB. (2003) 512–523
6. Xiong, X., Mokbel, M.F., Aref, W.G.: Sea-cnn: Scalable processing of continuous  $k$ -nearest neighbor queries in spatio-temporal databases. In: ICDE. (2005) 643–654
7. Yu, X., Pu, K.Q., Koudas, N.: Monitoring  $k$ -nearest neighbor queries over moving objects. In: ICDE. (2005) 631–642

8. Hu, H., Xu, J., Lee, D.L.: A generic framework for monitoring continuous spatial queries over moving objects. In: SIGMOD. (2005) 479–490
9. Mouratidis, K., Hadjieleftheriou, M., Papadias, D.: Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In: SIGMOD. (2005) 634–645
10. Brinkhoff, T.: A framework for generating network-based moving objects. *GeoInformatica* **6**(2) (2002) 153–180

# Optimizing Multiple In-Network Aggregate Queries in Wireless Sensor Networks\*

Huei-You Yang, Wen-Chih Peng, and Chia-Hao Lo

Department of Computer Science

National Chiao Tung University

Hsinchu, Taiwan, ROC

{hyyang, wcpeng, chlo}@cs.nctu.edu.tw

**Abstract.** In this paper, we explore the feature of sharing partial results of multiple queries to reduce the total number of messages incurred. Those queries sharing their partial results are referred to as backbones. Given a set of queries, we shall determine backbones with the purpose of minimizing the total number of messages. Specifically, given a set of queries, we derive a graph, where each vertex represents one query and the corresponding weight edge denotes the number of messages reduced by sharing partial results. Then, we develop a heuristic algorithm SB (standing for Selecting Backbones) to derive a cut in which both backbones and non-backbones are determined. Simulation results show that by sharing partial results, algorithm SB is able to significantly reduce the total number of messages involved.

## 1 Introduction

In monitoring applications of wireless sensor networks, queries are typically long-running and executed over a specified period. Since each query is independently performed, wireless sensor networks consume a considerable amount of energy when the number of queries increases. Queries issued could be categorized according to their aggregate operator and monitored time period. Queries in the same group have the same aggregate operator and monitored time period. As such, queries in the same group are able to further share their query results to reduce the number of messages. Given a set of queries with the same aggregate operator and time duration, we elaborate on sharing query results to reduce the total number of messages without influencing final query results. Queries are divided into two sets: backbone and non-backbone sets. Queries in the backbone set are issued as usual and should share their partial results with those queries in the non-backbone set. The problem addressed in this paper is that given a set of query trees, we should determine which query tree should be put in the backbone set and non-backbone set so as to minimize the total number of messages involved in multiple queries.

---

\* This paper is supported in part by the National Science Council, Project No. NSC 95-2221-E-009-061-MY3 and NSC 995-2221-E-009-026, Taiwan, Republic of China.

In order to determine which query tree should be put in the backbone set and the number of backbones, we first formulate the problem of selecting backbones and transform this problem into Max-Cut problem. Specifically, given a set of queries, we derive a graph, where each vertex represents one query and the corresponding weight edge denotes the number of messages reduced by sharing partial results. According to the graph derived, we develop a heuristic algorithm SB (standing for Selecting Backbones) to derive a cut in which both backbones and non-backbones are determined. Performance of algorithm SB is comparatively analyzed and simulation results show that by sharing partial results, algorithm SB is able to significantly reduce the total number of messages.

A significant amount of research efforts have been elaborated upon issues of in-network query processing for power saving in wireless sensor networks [3][5][6]. Prior works [2][6] explore the feature of in-network aggregation in which sensor nodes in a routing tree are able to perform aggregate operators. The authors in [1] proposed in-network materialized view that could be shared by multiple queries to reduce the number of messages. To the best of our knowledge, no prior works exploit the feature of sharing partial results of in-network aggregate queries, let alone formulating the problem of selecting backbones and devising algorithms to determine backbones for partial result sharing.

The rest of this paper is organized as follows. Preliminaries are presented in Section 2. In Section 3, we develop algorithm SB for backbone selection. Performance studies are conducted in Section 4. This paper concludes with Section 5.

## 2 Preliminaries

The goal of this study is to reduce the total number of messages spent for multiple queries. In order to share partial results, queries with the same aggregate operator and time duration are considered. Similar to prior works in [6], query  $Q_i$  is able to represent as a query tree, denoted as  $T_i$ . The leaf nodes of a query tree are data sources that will report sensing data and intermediate nodes of the query tree are used to aggregate sensing data from their child nodes. Hereafter, to facilitate the presentation of our paper, query  $Q_i$  is referred to as a query tree  $T_i$ . The number of messages spent for a query  $Q_i$ , expressed by  $N(T_i)$ , is the number of tree edges in  $T_i$ . For query tree  $T_i$ ,  $D_i(S_j)$  represents the partial result generated at sensor  $S_j$  and  $N_i(S_j)$  is the number of messages spent for partial result  $D_i(S_j)$ , which is the number of tree edges of the subtree rooted at sensor  $S_j$ .

To facilitate the presentation of this paper, the backbone set (respectively, the non-backbone set) is expressed by  $B$  (respectively,  $NB$ ). Clearly, by sharing partial results from backbones, non-backbones are able to reduce a considerable amount of messages. Denote the number of messages reduced for non-backbone  $T_j$  as  $R(T_j, B)$ . Thus, the total number of messages involved for a set of query trees can be formulated as follows:

$$\sum_{T_i \in B} N(T_i) + \sum_{T_j \in NB} (N(T_j) - R(T_j, B))$$

From the above formula, we could verify that minimizing the total number of messages is achieved by maximizing the number of messages reduced for queries in the non-backbone set. Intuitively, this problem is able to model as a Max-Cut problem. Explicitly, each query tree is viewed as a vertex and an edge represents the number of reduced messages achieved by sharing partial results.

### 3 Algorithm SB: Selecting Backbones

#### 3.1 Determining Edges and Weights Among Query Trees

When two query trees have some overlaps in their data sources, an edge will be added in the graph to represent the partial result sharing relationship. Specifically, suppose that the partial result on  $S_m$  of  $T_i$  is the same as the one on  $S_n$  of  $T_j$ . In other words,  $D_i(S_m)$  is the same as  $D_j(S_n)$  due to the same data sources. For query trees  $T_i$ ,  $T_j$ ,  $w_{i,j}(S_m, S_n)$  denotes the number of messages reduced when sensor  $S_n$  in  $T_j$  obtains the partial result of sensor  $S_m$  in  $T_i$ . To formulate the value of  $w_{i,j}(S_m, S_n)$ , we should consider that  $S_n$  should access the partial result  $D_i(S_m)$  from  $S_m$ . Therefore, an extra transmission cost is required and this extra transmission cost is therefore estimated as the minimal hop count between  $S_m$  and  $S_n$ , denoted as  $d_S(S_m, S_n)$ . Consequently, the value of  $w_{i,j}(S_m, S_n)$  is formulated as  $N_j(S_n) - d_S(S_m, S_n)$ .

To facilitate the presentation of all possible ways for sharing partial results from  $T_i$  and  $T_j$ ,  $W_{i,j}$  is used to represent the set of weights for various partial result sharing scenarios between  $T_i$  and  $T_j$ . As mentioned above, the weight of each possible sharing scenario is in fact in the form of  $w_{i,j}(S_m, S_n)$ , meaning that  $T_i$  shares the partial result in  $S_m$  to sensor  $S_n$  in  $T_j$ .

#### 3.2 Design of Algorithm SB

The objective of algorithm SB is to maximize  $\sum_{T_j \in NB} R(T_j, B)$ . Thus, algorithm SB is greedy in nature and selects the backbone with the maximal number of messages reduced for those non-backbones each time until no any message could be reduced when additional backbone is selected.

In essence, the value of  $R(T_j, B)$  is related to how the sensor nodes of  $T_j$  access partial results from backbones. Since there are many possible scenarios for  $T_j$  to get partial results from backbones, we should avoid redundant message transmissions when formulating the value of  $R(T_j, B)$ . Thus, we have the following property.

**Property 1.** *If non-backbone  $T_j$  gets a partial result for a node  $S_x$  in  $T_j$ , it is unnecessary to further access partial results for the ancestors or descendants of  $S_x$  in  $T_j$ .*

Assume that nodes  $S_y$  and  $S_z$  are the child nodes of node  $S_x$  and both nodes  $S_y$  and  $S_z$  access partial results from backbones. Obviously, since the partial results of  $S_y$  and  $S_z$  is used to aggregate the result on  $S_x$ , node  $S_x$  should not access the partial result from backbones. For the same reason, it is also unnecessary to get partial results for the descendants of  $S_y$  or  $S_z$ .

In light of Property 1, we have developed a procedure to determine how many messages could be reduced through the partial result sharing. The algorithmic form of the proposed procedure is given below:

Procedure  $R(T_j, B)$ :

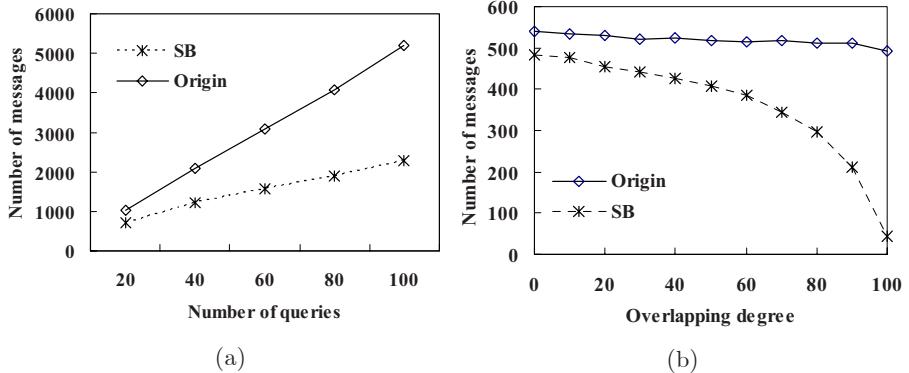
1. set  $Y = \cup_{i \in B} W_{i,j}$ . To, to determine the union set of sensors from the auxiliary table ;
2. Generate the power set of  $Y$ , denoted as  $P(Y)$ , is the set of all subsets of  $Y$ ;
3.  $\forall X \in P(Y)$ , if there exists any ancestor or descendant relationship in  $X$ , prune  $X$  from  $P(Y)$ ;
4. return  $\max_{\forall X \in P(Y)} (\sum_{S_n \in X \text{ and } i \in B} w_{i,j}(S_m, S_n))$

In the beginning, we will determine the set of  $Y$  from the auxiliary table. As described above, the auxiliary table will contain all the detailed information related to the partial result sharing. Thus, given the backbone set, we could easily decide the set of  $Y$ . In fact,  $Y$  contains all the sensors in  $T_j$  that could access the partial results from backbones. In order to enumerate all the possible scenarios, we should generate the power set of  $Y$ , denoted as  $P(Y)$ . According to Property 1, we should avoid redundant message cost and thus, for each set in  $P(Y)$ , we should check whether there is any ancestor and descendant relationship or not. Note that one could refer to query tree  $T_j$  to verify any ancestor and descendant relationship. As such, the set of  $P(Y)$  has all the possible scenarios of partial result sharing for  $T_j$ . Consequently, the number of messages reduced for  $T_j$  is able to be the maximal value among these possible scenarios.

To evaluate the benefits of selecting query tree  $T_i$  as a backbone, we have the following definition.

**Definition 1.** The *backbone gain* achieved by selecting  $T_i$  as a backbone, denoted by  $\delta(T_i)$ , can be formulated as  $\delta(T_i) = \sum_{T_j \in (NB - T_i)} R(T_j, B \cup T_i) - \sum_{T_j \in NB} R(T_j, B)$ .

In light of Definition 1, we propose a heuristic algorithm SB that iteratively select backbones according to *backbone gains* of query trees. Initially, the backbone set is empty and the non-backbone set is the set of query trees given. For each query tree in the non-backbone set, we will calculate the corresponding backbone gain. Then, the query tree with the maximal backbone gain is included in the backbone set. Once one query tree is selected as a backbone, we should update backbone gains for query trees in the non-backbone set. Similarly, according to the backbone gains of query trees in the non-backbone set, we will select the one with the maximal backbone gain as a backbone. Algorithm SB selects the



**Fig. 1.** Performance comparison of Origin and SB (a) with number of queries varied. (b) with overlapping degree varied.

query trees in the non-backbone set iteratively until no additional query tree is selected in the backbone set. When query trees in the non-backbone set have their corresponding backbone gains smaller than zero, no query tree will be selected in the backbone set since no more benefit will be earned. As such, a set of query trees is divided into two sets: the backbone set and the non-backbone set, which is akin to Max-Cut problem with the objective of maximizing the cut, meaning that the number of messages reduced is maximized.

## 4 Performance Evaluation

### 4.1 Simulation Model

A wireless sensor network is simulated, where there are 500 sensors randomly deployed in a  $500 \times 500 m^2$  region. The sink is at the left-top corner of the region. The transmission range of sensors is set to 50 m. Users submit queries to the sink and each query utilizes scheme TAG [2] to form a query tree, where the root node is the sink. Query range is referred to those sensors whose sensing data are the data sources of one query tree. A query region set of a query tree is referred to the set of nodes in the query trees except the root node (i.e., sink). Assume that two query trees with their query region sets as  $R_1$  and  $R_2$ . Then, we define the overlapping degrees of these two query trees as  $\frac{R_1 \cap R_2}{R_1 \cup R_2}$ . Note that with higher value of overlapping degree, query trees have more sensors in their overlap area, which means that more partial results are sharable among query trees. For the comparison purpose, scheme *Origin* is referred to the scenario that queries are performed as usual without any partial result sharing.

### 4.2 Experimental Results

First, we investigate the impact of sharing partial query results, where the overlapping degree is set to 50% and we set the query range to  $100 \times 100 m^2$ .

As can be seen in Fig. II(a), the numbers of messages of scheme Origin, algorithm SB increase as the number of queries increases. Note that through the partial result sharing, algorithm SB has smaller numbers of messages involved. Note that when query trees have more overlapping area of query regions, these query trees are likely to have more opportunities to share partial results. Now, we examine the impact of overlapping degree, where the number of queries is set to 10 and the query range of each query is set to  $100 \times 100 m^2$ . The performance of Origin and SB with the overlapping degree varied is shown in Fig. II(b). The number of messages is reduced in SB as the overlapping degree increases. This phenomenon agrees with our above statement that with a larger value of the overlapping degree, query trees have more changes to share partial results. As a result, the performance of SB is better than that of Origin.

## 5 Conclusion

In this paper, we exploited the feature of sharing partial results to reduce the total number of messages. Specifically, given a set of queries, we derived a graph, where each vertex represents one query and the corresponding weight edge denotes the number of messages reduced by sharing the partial results. According to the graph derived, we developed heuristic algorithm SB to derive a cut in which both backbones and non-backbones are determined. Performance of algorithm SB was comparatively analyzed and experimental results show that by sharing the partial results, algorithm SB is able to significantly reduce the total number of messages.

## References

1. K. C. K. Lee, W.-C. Lee, B. Zheng, and J. Winter. Processing multiple aggregation queries in geo-sensor networks. In Proceeding of the 11th International Conference on Database Systems for Advanced Applications (DASFAA), pages 2034, 2006.
2. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. ACM SIGOPS Operating Systems Review, 36(SI):131146, 2002.
3. S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. ACM Transactions on Data Base Systems (TODS), 30(1):122173, 2005.
4. A. Sharaf, J. Beaver, A. Labrinidis, and K. Chrysanthis. Balancing energy efficiency and quality of aggregate data in sensor networks. The VLDB Journal, 13(4):384–403, 2004.
5. N. Trigoni, Y. Yao, A. J. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. In Proceeding of the first IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), pages 307321, 2005.
6. Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. SIGMOD Record, 31(3):918, 2002.

# Visible Nearest Neighbor Queries

Sarana Nutanong<sup>1</sup>, Egemen Tanin<sup>1,2</sup>, and Rui Zhang<sup>1</sup>

<sup>1</sup> Department of Computer Science and Software Engineering,

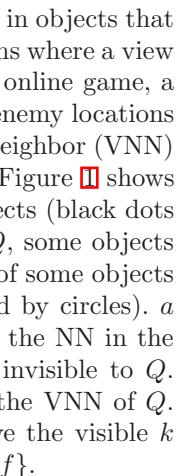
University of Melbourne, Victoria, Australia

<sup>2</sup> NICTA Victoria Laboratory, Australia

**Abstract.** We introduce the visible  $k$  nearest neighbor (V $k$ NN) query, which finds the  $k$  nearest objects that are visible to a query point. We also propose an algorithm to efficiently process the V $k$ NN query. We compute the visible neighbors incrementally as we enlarge the search space. Our algorithm dramatically reduces the search cost compared to existing methods that require the computation of the visibility of all objects in advance. With extensive experiments, we show that our algorithm to process the V $k$ NN query outperform the existing algorithms significantly.

**Keywords:** Nearest Neighbor, Spatial Algorithms, Spatial Data Structures.

## 1 Introduction

In many interactive spatial applications, users are only interested in objects that are visible to them. For example, tourists are interested in locations where a view of a scene, e.g., sea or mountains, is available; in an interactive online game, a player is commonly interested in having an overview map of the enemy locations that can be seen from his/her position. A simple visible nearest neighbor (VNN) query finds the nearest object that is visible to a query point. Figure  shows an example of the VNN query. The dataset consists of data objects (black dots and circles) and obstacles (lines).  $Q$  is the query point. From  $Q$ , some objects are visible ( $b, d, f, h$ , represented by black dots) while the views of some objects are blocked by obstacles, namely, invisible ( $a, c, e, g$ , represented by circles).  $a$  has the smallest distance to  $Q$  among all objects, therefore  $a$  is the NN in the traditional sense. However,  $a$  is not the VNN of  $Q$  since  $a$  is invisible to  $Q$ . Among all the visible objects,  $b$  is nearest to  $Q$ , therefore  $b$  is the VNN of  $Q$ . In analogy to the  $k$  nearest neighbor ( $k$ NN) query, we can have the visible  $k$  nearest neighbor query. In the example, the V3NN of  $Q$  is  $\{b, d, f\}$ .

Formally, the VNN query is defined as follows:

**Definition 1 (Visible nearest neighbor (VNN) Query).** Given a data set  $S$  and a query point  $Q$ , find an object  $O \in S$ , so that: (1)  $O$  is visible to  $Q$ ; and (2)  $\forall O' \in S$ , if  $O'$  is visible to  $Q$ , then  $\text{dist}(O, Q) \leq \text{dist}(O', Q)$ , where  $\text{dist}()$  is a function to return the distance  between the query point and an object.  $O$  is called the visible nearest neighbor (VNN) of  $Q$ .

<sup>1</sup> In this paper, we focus on the Euclidian distance, although any distance function can be used in general.

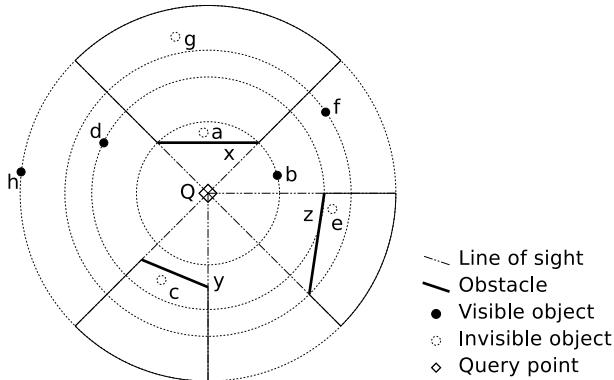
Further, the definition of the VkNN query is given as follows:

**Definition 2 (Visible  $k$  nearest neighbor (VkNN) Query).** Given a data set  $S$  and a query point  $Q$ , find a set of objects  $A$ , so that: (1)  $A$  contains  $k$  objects from  $S$ ; and (2)  $\forall O \in A$ ,  $O$  is visible to  $Q$ ; and (3)  $\forall O' \in S - A$ , if  $O'$  is visible to  $Q$ , then  $\text{dist}(O, Q) \leq \text{dist}(O', Q)$ , where  $\text{dist}()$  is a function to return the distance between the query point and an object.  $A$  is called the visible  $k$  nearest neighbor set (VkNN) of  $Q$ .

The notion of visibility has been extensively studied in the area of computer graphics and computational geometry [1]. Specifically, given a set of spatial objects and obstacles, there are efficient algorithms to determine the visible regions, i.e., the regions where objects are visible. Therefore, a naive method to process the VkNN query is to use a NN search algorithm to find objects progressively according to their distances to the query point and use the visibility knowledge as the post-condition to discard the invisible objects. The process stops when  $k$  visible objects are retrieved. But it requires the visible regions to be determined in advance, which accesses all the obstacles.

In this paper, we propose an algorithm for VkNN search without pre-computing the visible regions. Our algorithm is based on the observation that a farther object cannot effect the visibility of a nearer object. Therefore, we can start from retrieving the nearest object and incrementally obtain the knowledge of visibility while finding visible neighbors. By this means, the cost for determining the visibility of data objects is minimized.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes our proposed VkNN algorithm. Section 4 presents the results of our experimental study and Section 5 concludes the paper.



**Fig. 1.** The VNN query

## 2 Related Work

An incremental nearest neighbor algorithm was introduced in [6] based on a spatial index such as the R-tree [5]. Using this incremental approach, the number of

nearest neighbors required,  $k$ , need not to be specified in advance and can be retrieved in order incrementally. The major benefit of this algorithm compared to the kNN approach in [8] is that obtaining the next nearest neighbor after having obtained the  $k$  nearest neighbors incurs minimal additional effort. The algorithm uses a priority queue to maintain the candidates (index nodes or data entries) ranked by an optimistic distance estimator. Every time an object is retrieved from the priority queue, the property of the optimistic distance estimator ensures that nothing left in the priority queue can be nearer to the query point than the object. Therefore, the objects are retrieved incrementally in increasing order of their distances to the query point. Our VkNN uses the same incremental search strategy to find the results. More importantly, we incrementally compute the visibility knowledge, therefore the visibility determination cost is greatly reduced.

The notion of the constrained nearest neighbor query which combines the NN problem with the linear-constrained-search problem is introduced in [4]. A constrained nearest neighbor query finds the nearest neighbor inside a convex-polygonal spatial constraint. Only nearest neighbors satisfying the constraint set are returned and only regions satisfying the constraints are explored. Although the visibility knowledge can be modeled as a set of constraints, it is inefficient to use the constraint nearest neighbor algorithm to solve the VkNN problem. This is because the constraint nearest neighbor query requires the constraints to be given in advance causing preprocessing of the visibility constraints.

Recently, a new type of spatial query, nearest surround (NS), is presented in [7]. A nearest surround query finds the nearest object for each distinct range of angles around the query point. Each returned object is associated with its orientation, which is the range of angles in which the object is the nearest to the query point. By this means, only objects that are not entirely eclipsed by nearer objects can be returned as results. This is similar to VkNN which finds  $k$  visible objects around the query points. The main difference of these two spatial queries is that NS finds all “visible” objects around the query point whereas the number of objects in VkNN is user-determined. Using our *pre-pruning* mechanism, we can modify VkNN to efficiently find the complete set of nearest surrounders by running VkNN to completion and associating each returned object with its visible range of angles.

### 3 Visible Nearest Neighbor Algorithms

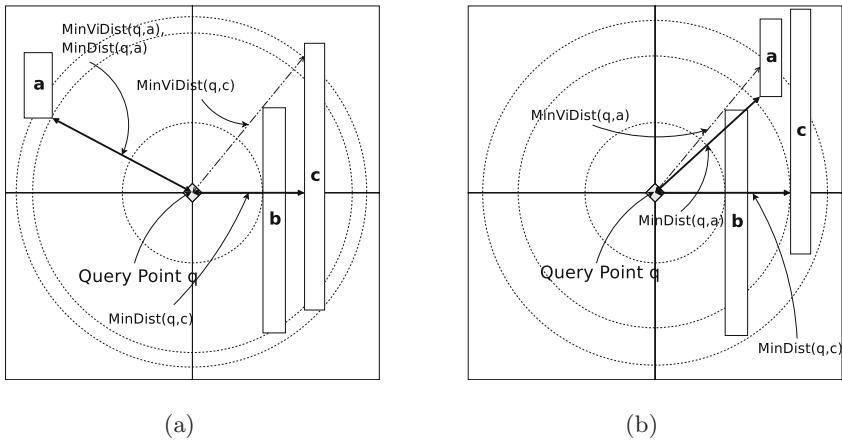
Our algorithm is based on the observation that a farther object cannot effect the visibility of a nearer object. Therefore, both results and visibility knowledge can be incrementally obtained. By doing this, the cost for determining the visibility of data objects is minimized.

Without loss of generality, to simplify our discussion, we make no distinction between the point objects which can be returned as results and the obstacles that create invisible regions of space; we refer to both of them as *objects*. In general, objects are represented by polygons (points are special cases of polygons). Objects represented as polygons (i.e., with extents) can be partially visible.

Therefore, we introduce a new distance function, called *MinViDist*, which returns the distance between a query point and the nearest visible point of an object with regard to a given visibility setting. This distance function is different from the commonly used MinDist function. The MinDist between a polygon and a point is the distance between the point and the nearest point in the polygon.

Formally, the MinViDist function is defined as follows:

**Definition 3 (MinViDist).** Given a polygon  $P$  and a query point  $Q$ , MinViDist is the distance between  $Q$  and a point  $T \in P$ , so that (1)  $T$  is visible to  $Q$ ; and (2)  $\forall T' \in P$ , if  $T'$  is visible to  $Q$ , then  $dist(T, Q) \leq dist(T', Q)$ , where  $dist()$  is a function to return the distance between two points.



**Fig. 2.** Comparison of MinDist and MinViDist

As shown in Fig. 2(a), the MinViDist between the query point  $q$  and the object  $c$  is the length of the dashed line pointed to the surface of  $c$ , whereas the MinDist between  $q$  and  $c$  is the length of the solid line pointed to the surface of  $c$ , passing through the obstacle  $b$ . The MinDist and MinViDist for object  $a$  are the same since no obstacle is in between. If we order the objects according to MinDist, we get  $\{b, c, a\}$ ; if we order the objects according to MinViDist, we get  $\{b, a, c\}$ . Therefore, if we rank the objects according to MinDist, we may not get the answers for a V<sub>k</sub>NN query in the correct order.

Figure 2(b) shows another example of the difference between MinDist and MinViDist. The MinViDist between the query point  $q$  and the object  $a$  is the length of the dashed line pointed to the surface of  $a$ , whereas the MinDist between  $q$  and  $a$  is the length of the solid line pointed to the surface of  $a$ . They are different because of the obstacle  $b$ . The MinDist between  $q$  and  $c$  is the length of the solid line pointed to the surface of  $c$ , passing through the obstacle  $b$ ; whereas the MinViDist between  $q$  and  $c$  is infinity, that is,  $c$  is invisible to  $q$ . If we issue a 3NN query (which actually uses MinDist to rank the objects), we get  $\{b, c, a\}$ ; if we issue a V3NN query (which actually uses MinViDist to rank the

objects), we get  $\{b, a\}$ .  $c$  is not returned for V3NN since it is invisible. Therefore, using MinDist, we may not even get the same set of answers as those retrieved according to MinViDist.

Given the definition of MinViDist, we can now describe our algorithm to process the VkNN query. In our presentation, we assume that all the objects are indexed in an R-tree [5], although our algorithms are applicable to any hierarchical spatial index structure such as the k-d-tree [3] or the quadtree [9]. We propose three variations of the algorithm which differ in whether pruning objects by visibility is done before or after retrieving a node, and differ in what distance estimator we use to order objects in the priority queue that maintains the candidates. The results are ranked according to the MinViDist function for all of three variations.

```

ALGORITHM PrePruning-MinDist (IsNewQuery, QueryPoint)
1 if IsNewQuery then
2 PriorityQueue \leftarrow PriorityQueueCreate()
3 VisibilityConstraintSet \leftarrow VisibilityConstraintSetCreate(QueryPoint)
4 NewPQueueNode \leftarrow PQueueNodeCreate(RootNode)
5 PriorityQueue.Insert(NewPQueueNode)
6 endif
7 while PriorityQueue.IsEmpty() = false do
8 PQueueNode \leftarrow PriorityQueue.Front()
9 SpatialEntity \leftarrow PQueueNode.SpatialEntity
10 PriorityQueue.PopFront()
11 if IsInvisible(VisibilityConstraintSet, SpatialEntity) then
12 continue
13 else if IsObject(SpatialEntity) then
14 PQueueNode.Distance \leftarrow MinViDist(VisibilityConstraintSet, QueryPoint, SpatialEntity)
15 if PQueueNode.Distance < PriorityQueue.Front().Distance then
16 Update(VisibilityConstraintSet, SpatialEntity)
17 return (SpatialEntity, PQueueNode.Distance)
18 else
19 PriorityQueue.Insert(PQueueNode)
20 endif
21 else if IsBlock(SpatialEntity) then
22 for SubEntity in SpatialEntity.SubEntities do
23 if IsVisible(VisibilityConstraintSet, SpatialEntity) then
24 NewPQueueNode \leftarrow NewPQueueNodeCreate()
25 NewPQueueNode.Distance \leftarrow MinDist(QueryPoint, SpatialEntity)
26 PriorityQueue.Insert(NewPQueueNode)
27 endif
28 endfor
29 endif
30 endwhile
31 return nil

```

**Fig. 3.** Algorithm PrePruning-MinDist

**PostPruning:** We use a similar kNN search algorithm as in [6] (MinDist is used to sort the candidates in the priority queue). The only differences are the post-pruning process discarding invisible objects and result-ranking use MinViDist. This variation of the algorithm is an adaptation of the existing algorithm [6].

**PrePruning-MinDist:** Similar to PostPruning, MinDist is used as the estimator but the index nodes and objects are checked for visibility before they are retrieved. By this means, we avoid needlessly searching invisible regions

which will not produce useful results. Figure 3 shows the detailed steps of the algorithm. The algorithm is also based on the incremental NN algorithm [6]. The differences between the two algorithms are that: (1) index nodes and objects are checked for visibility before they are retrieved from the R-tree (Line 23) and after they are dequeued from the head of the priority queue (Line 11); and (2) results are ranked according to MinViDist (Line 14).

**PrePruning-MinViDist:** This variation differs from PrePruning-MinDist only in that we use a MinViDist (which is more accurate and more expensive to calculate) as the metric to order the candidates in the candidate priority queue. This is done by replacing MinDist in Fig. 3 Line 25 with MinViDist.

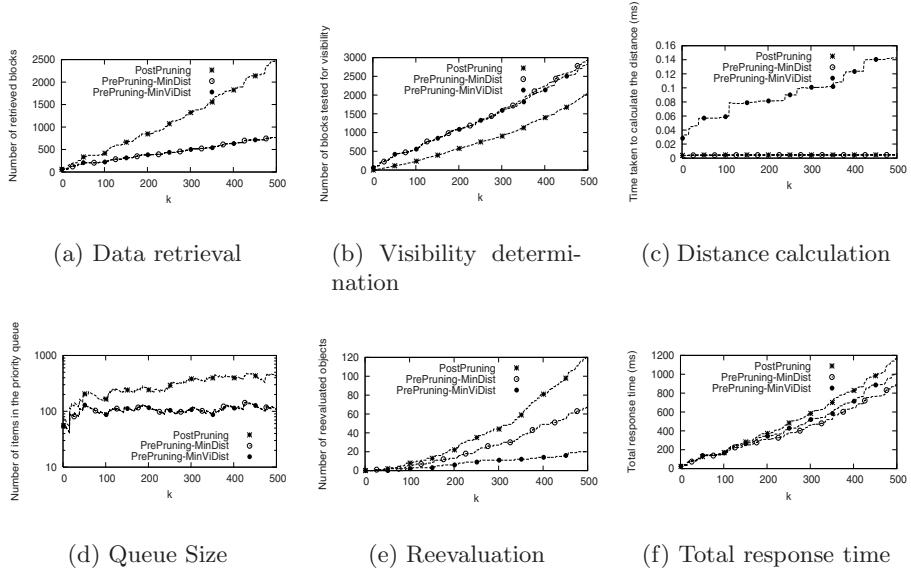
## 4 Performance Evaluation

This section compares the performance of the three variations of the VkNN search algorithm described in Section 3. In our implementation, a disk-based R\*-tree [2] is used. The data set contains 10,000 objects synthetically generated and uniformly distributed in a unit 2-dimensional space. These 10,000 objects also serve as obstacles. The width and height of each object are randomly generated in a uniform manner ranging between 0.0001 and 0.001 units. The fanout of the R\*-tree nodes is 24. The performance evaluation is conducted on a Intel Pentium 4 machine with the main memory of 2 GB.

The cost for VkNN calculation can be broken down into five components: data retrieval, visibility determination, distance calculation, priority queue access, and reevaluation (the cost for reinserting objects back into the priority queue for reevaluation). This breakdown can be used to determine which of the three approaches presented in Section 3 is more suitable in different settings.

The experimental results are presented in six charts. Each chart plots the performance of the three approaches as a function of  $k$ . Figure 4(a) compares the data-retrieval costs of the three variations, which is measured by the number of blocks accessed. The result for visibility-determination costs is given in Fig. 4(b). Figure 4(c) shows the time to calculate the distance of a node in microseconds. Fig. 4(d) presents the maintenance cost of the priority queue in terms of the queue size. It is plotted on the logarithmic scale because the priority-queue access cost is logarithmic with respect to the queue size (when implemented using heaps). Figure 4(e) displays the reevaluation cost measured by the number of objects reinserted into the priority queue. This incurs a different type of cost from the data retrieval cost because it does not involve storage access. The total response time is displayed in Fig. 4(f).

These experimental results suggest that PrePruning-MinDist have the same data-retrieval and priority-queue maintenance costs as PrePruning-MinViDist but the time taken to calculate the distance of each block for PostPruning and PrePruning-MinDist is much smaller (the reevaluated object counts are small in general for our settings and can be ignored). Due to the fact that PostPruning has a larger search space, it has higher reevaluation and queue processing costs. According to the results, the improvement in terms of search-space reduction

**Fig. 4.** Experimental Results

made by using MinViDist instead of MinDist as the distance estimator is considered to be insignificant while the cost for MinViDist calculation is much higher than MinDist calculation. We suspect the reason that PrePruning-MinDist and PrePruning-MinViDist incur the same amount of search space is that the R\*-tree margin-minimization criterion [2] prefers quadratic-shaped index nodes. This means that most nodes are either invisible or have MinDist about the same as MinViDist; instead of having a large difference in MinDist and MinViDist. We can therefore conclude that PrePruning-MinDist which uses the MinDist metric as the distance estimator and visibility pruning is a better approach for VNN. The MinViDist metric should be primarily considered for ranking purposes than as a search-estimator. In disk-based R\*-trees, PrePruning-MinDist is clearly a better option than PostPruning. Since in such settings, the data-retrieval costs dominates the total cost of query processing [6]. A setting that would make the first approach comparable to PrePruning-MinDist is when the R\*-tree is memory-based and the CPU speeds are very slow, because this makes it cheaper to retrieve a block from the R\*-tree but more expensive to determine the visibility of a block.

## 5 Conclusion

In this paper, we introduced a new type query, the *visible k nearest neighbor* (V<sub>k</sub>NN) query. We also introduced a new metric called *minimum visible distance* (MinViDist) for result ranking as well as search ordering. Furthermore, we propose an algorithm (particularly, three variations of the algorithm) to

process the VkNN query. All the three variations build up the visibility knowledge incrementally as the visible nearest objects are retrieved and therefore the computation cost for visibility determination is minimized. It is shown in the experimental results that the latter two variations, PrePruning-MinDist and PrePruning-MinViDist, put more effort on the visibility pruning in order to reduce the data-retrieval cost. This could be beneficial in settings with disk-based or network-based storage where the data-retrieval costs are more critical than the CPU costs. These two variations differ in the computation cost of calculating the distance estimator and the number of disk accesses for visible objects, which is a tradeoff depending on computing power and object retrieval cost. Both of them are more efficient than the first variation, PostPruning, which is an adaptation of the existing algorithm [6]. In our experiments, the improvement in terms of response time of the query processing is up to 35%.

## References

1. T. Asano, S. K. Ghosh, and T. C. Shermer. *Visibility in the plane*, pages 829–876. Handbook of Computation Geometry. Elsevier Science Publishers, Amsterdam, The Netherlands, 2000.
2. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R\*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD Conf.*, pages 322–331, Atlantic City, NJ, USA, 1990. ACM Press.
3. J. L. Bentley. Multidimensional binary search trees used for associative searching. *CACM*, 18(9):509–517, 1975.
4. H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. El Abbadi. Constrained nearest neighbor queries. In *Proceedings of the SSTD Conf.*, pages 257–278, London, UK, 2001. Springer-Verlag.
5. A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conf.*, pages 47–57, Boston, MA, USA, 1984. ACM Press.
6. G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999.
7. K. C. K. Lee, W. C. Lee, and H. V. Leong. Nearest surround queries. In *Proceedings of the ICDE Conf.*, pages 85–94, Atlanta, GA, USA, 2006. IEEE Computer Society.
8. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD Conf.*, pages 71–79, San Jose, CA, USA, 1995. ACM Press.
9. H. Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, 1984.

# On Query Processing Considering Energy Consumption for Broadcast Database Systems

Shinya Kitajima<sup>1</sup>, Jing Cai<sup>1</sup>, Tsutomu Terada<sup>2</sup>,  
Takahiro Hara<sup>1</sup>, and Shojiro Nishio<sup>1</sup>

<sup>1</sup> Dept. of Multimedia Eng., Graduate School of Information Science and Technology,  
Osaka University, 1-5 Yamadaoka, Suita, Osaka 565-0871, Japan

{kitajima.shinya, cai, hara, nishio}@ist.osaka-u.ac.jp

<sup>2</sup> Cybercommunity Division, Cybermedia Center, Osaka University

5-1 Mihogaoka, Ibaraki, Osaka 567-0047, Japan

tsutomu@cmc.osaka-u.ac.jp

**Abstract.** In recent years, there has been an increasing interest in the broadcast database system where the server periodically broadcasts contents in a database to mobile clients such as portable computers and PDAs. There are three query processing methods in the broadcast database system, while each method consumes different amount of power for query processing. In this paper, we propose a new query processing method which dynamically chooses an appropriate method among the three query processing methods by considering energy consumption.

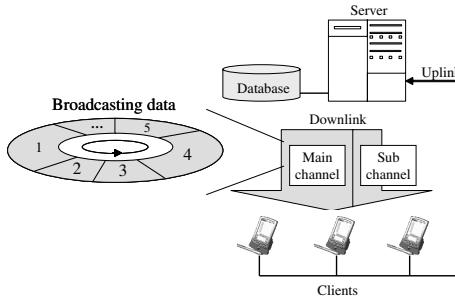
**Keywords:** data broadcast, broadcast database system, query processing, power consumption.

## 1 Introduction

The recent evolution of wireless communication technologies has led to an increasing interest in broadcast information systems in which data is disseminated via the broadcast. In such systems, a server broadcasts various data periodically via the broadband channel, while clients pick out and store necessary data. There are many studies for improving the performance of broadcast information systems [1][2][3]. Most of them deal with broadcast data as data items simply, and do not address the performance improvement by considering contents and characteristics of broadcast data.

In this paper, we assume a broadcast system that the server periodically broadcasts contents in a database and clients issue queries to retrieve data from the database. We call such a system *broadcast database system*. There are three basic query processing methods in this system. However, the performance of each method changes according to the system situation such as query frequency.

In [5], we proposed a query processing method which chooses the method with the least response time among these three methods. In this paper, based on the method in [5], we propose a new query processing method which considers the energy consumption of mobile clients when choosing a query processing method.



**Fig. 1.** Broadcast database system

Furthermore, the simulation evaluation confirms that the proposed method improves the lifetimes of clients with a low battery power left.

The remainder of this paper is organized as follows. Section 2 describes the outline of a broadcast database system and introduces three basic query processing methods and the traditional method in the broadcast database system. Section 3 explains our method in details. Section 4 evaluates the performance of our method. Finally, we conclude the paper in Section 5.

## 2 Broadcast Database System

Fig. 1 illustrates the concept of a broadcast database system. In this system, the server broadcasts contents in a relational database via the broadcast channel and processes queries from clients. Clients issue queries to retrieve the necessary data from the database. Clients have a small storage, low power resource, and low CPU capability, such as a PDA. The broadcast channel from the server to clients is divided into two channels: a broadband *main channel* to disseminate the contents in a database repeatedly, and a narrowband *sub channel* to disseminate the other data. Moreover, there is a narrowband uplink channel from clients to the server. Clients use the uplink channel to send queries to the server.

### 2.1 Assumed Environment

We assume that our method is used for disseminating information to many and unspecified users. For example, for an information service in a shopping center, the server broadcasts data in the database including advertising information, shop information, and goods information in the shopping center, while thousands of users receive the broadcast information and retrieve the necessary information. The broadcast data contains maps of shops and images of goods.

The users occasionally issue queries to the server to retrieve the information, such as a natural join operation “I want the image of item A, and the map to the shop selling the item”. We assume several minutes’ delay for receiving query result is acceptable for clients. On the other hand, users set the deadline of the response time to each query. When users cannot receive query result by the time of deadline, the query fails. Although users stay in shopping center for a certain

time, the battery is limited and the users who run out of the battery cannot receive the service.

## 2.2 Query Processing Methods

In the broadcast database system, there are three basic query processing methods and one adaptive method as follows.

**On-demand method:** A client sends a query to the server through the uplink. The server processes the query and broadcasts the query result via the sub channel. In this method, the query processing is completely done by the server, no workspace is required for query processing at client, thus the energy consumption is low.

**Client method:** A client stores all the tables related to the query, and processes the query by itself. Query processing causes a heavy workload on the client which consumes a lot of battery power.

**Collaborative method:** A client sends a query to the server through the uplink. The server processes the query, attaches the query identifier to the tuples that appear in the query result, creates rules for the client to process the data, and then broadcasts the rules via the sub channel. Based on the received rules, the client receives the necessary tuples via the main channel referring to the identifiers, and reconstructs the query result by combining these tuples<sup>[4]</sup>.

**LRT (Least Response Time) method:** The system performance, when each method is used individually, changes with the environmental conditions such as query frequency. In the LRT method<sup>[5]</sup>, when the server receives a query, it calculates the response time respectively for the on-demand method, the client method, and the collaborative method, and then chooses a query processing method with the least response time.

## 3 ELEC Method

In the LRT method, there is a problem that the lifetimes of clients with a low battery become short. Therefore, we propose a new query processing method, called ELEC (Extended LRT considering Energy Consumption) method, that considers the battery capability of mobile clients when choosing a query processing method, and improves the lifetimes of clients which remain a low battery power.

### 3.1 Outline

In the ELEC method, the server basically chooses a query processing method according to the LRT method. However, the server checks the remaining battery of the client issuing query, and preferentially chooses the method with the lowest energy consumption when the remaining battery is lower than the threshold  $P_{TH}$ . The procedure of the query processing is as follows.

1. If the remaining battery of the client which currently issues a query is more than  $P_{TH}$ , the server chooses the query processing method according to the LRT method.
2. Otherwise, the server chooses the query processing method with lowest energy consumption.

The optimal  $P_{TH}$  will be changed according to the query frequency, remaining battery of clients, and so on.

### 3.2 Calculation of the Thresholds

For deciding the optimal threshold  $P_{TH}$ , the server duplicates last  $q$  queries while changing the tentative value of the threshold. Then, the server sets the threshold, so that the energy consumption for clients of which the remaining battery is less than the threshold, becomes least.

1. We define the remaining battery of the client which issues the last  $x$ -th query  $Q_x$  as  $P_x$ . We also define the sequence of numbers which consists of  $q$  elements as  $A = (P_1, P_2, \dots, P_q)$ , and the ascending sequence of  $A$  as the sequence of nominated thresholds  $A'$ .
2. We represent the  $k$ -th element of  $A'$  as  $A'[k]$ . The server takes the following steps for  $A'[i](i = 1, 2, \dots, q)$ .
  - (2a) The server duplicates last  $q$  queries by setting  $A'[i]$  as tentative threshold.
  - (2b) The server calculates the energy consumption per query of which the remaining battery is less than  $A'[i]$ , which is represented as  $E_i$ .
  - (2c) If  $E_j > E_{j-1} > E_{j-2}(j = 1, 2, \dots, q)$ , the server judges the optimal threshold is less than  $A_j$ , and goes to step 3.
3. We represent the minimum of  $E_i(i = 1, 2, \dots, q)$  as  $E_{min\_i}$ , while the threshold for next  $q$  queries as  $A'[min\_i]$ .

## 4 Evaluation

This section evaluates the ELEC method. Three evaluation criteria are introduced as follows.

**Lifetime:** The elapsed time from the user arrival to the user exit, due to either spending enough time or running out of the battery. Note that clients consume battery only by the processes related to query processing.

**Success rate:** The ratio of the queries of which clients could get the results to all of the queries clients issued.

**Response time:** The average elapsed time from the query generation to the acquirement of the query result. Note that the response time does not include the time for transmitting a query from a client to the server and the time for processing the data at the client side or the server side, since they are adequately short.

**Table 1.** Parameters

| Parameter                                                 | Value                            |
|-----------------------------------------------------------|----------------------------------|
| Simulation time[sec]                                      | 36000                            |
| Estimated staying time of users[sec]                      | 7200                             |
| Query interval for a user[sec]                            | 300                              |
| Deadline of response time[sec]                            | 80                               |
| Number of tuples                                          | 10000                            |
| Size of a tuple[KByte]                                    | 10                               |
| Number of identifiers                                     | 200                              |
| Bandwidth of main/sub channel[Mbps]                       | 10/1                             |
| Size of a processing rule[KByte]                          | 1                                |
| Average ratio for all tuples of necessary tuples          | 0.003                            |
| Standard deviation for all tuples of necessary tuples     | 0.001                            |
| The initial remaining battery of each client[unit energy] | 100 - 1000(uniform distribution) |
| The storage capacity of each client[MB]                   | 1 - 100(uniform distribution)    |
| The speed of writing/reading data[MB/s]                   | 10/15                            |

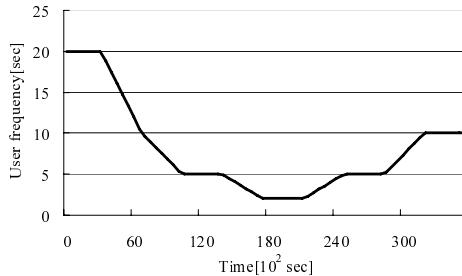
#### 4.1 Simulation Environment

In the evaluation, the database schema and the query model is supposed for an information service in a shopping center as described in Section 2.1. The database consists of a shop table and a goods table. For the sake of simplicity, all tuples are supposed to be the same size. Moreover, a client only issues queries of a natural join with the shop table and the goods table.

Table 1 shows the parameters used in the evaluation. The rate of necessary tuples represents the rate of tuples which are included in the query result to all tuples in the table. The user arrival intervals are given by the exponential distribution with a parameter of user arrival frequency, which is changed as shown in Fig. 2. Each user has a mobile client to access service. A user, who arrives at the shopping center, issues a query according to the query interval, with the deadline of the response time, the storage capacity, and the remaining battery. Each user exits the shopping center after elapsing the estimated staying time. We assume that the CPU capability of all clients are the same. Moreover, Mahesri et al's measurement results in [6] are used to define that the energy consumption of query processing is proportional to time  $t$ , it is presented as total sum of the energy consumption for CPU ( $E_C^h = 2t$  (when CPU load is high),  $E_C^l = 0.5t$  (when CPU load is low)), the energy consumption for wireless LAN ( $E_W = t$ ), and the energy consumption for I/O ( $E_I = t$  for reading,  $E_O = t$  for writing).

#### 4.2 Simulation Results

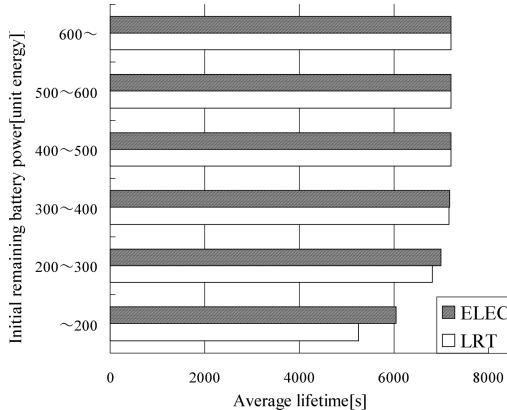
Table 2 shows the evaluation results of the success rate and the average response time of the LRT method and the ELEC method. Moreover, Fig. 3 shows the evaluation results of the average lifetime under different initial remaining battery of clients, which is classified in every 100 unit energies. The parameter  $q$  of the ELEC method is set to 50 according to the preliminary experiment.



**Fig. 2.** Change of user arrival frequency

**Table 2.** Success rate and response time

| Method | Success rate | Average response time |
|--------|--------------|-----------------------|
| LRT    | 94.4%        | 40.4[sec]             |
| ELEC   | 93.5%        | 41.9[sec]             |



**Fig. 3.** Average lifetime

Table 2 and Fig. 3 show that the average lifetime of the ELEC method is much longer than that of the LRT method, though the success rate and the average response time of the ELEC method are a little worse than those of the LRT method. In the ELEC method, the lifetime of clients which have a low remaining battery is long, since the server preferentially chooses the method that has lowest energy consumption when the remaining battery of the client is low. However, the restriction of the sub channel, storage, and identifier worsens the success rate and the average response time, due to continuously choosing the query processing method with the lowest energy consumption.

## 5 Conclusions

In this paper, we proposed a new query processing method which dynamically chooses a query processing method by considering the energy consumption. With the proposed method, the server preferentially chooses the method with the lowest energy consumption when the remaining battery is lower than the threshold. The simulation results confirmed that the proposed method improved the lifetime of the clients which have a low remaining battery.

In future, we plan to examine a method which decides the threshold according to the CPU capability of clients.

**Acknowledgments.** This research was partially supported by The 21st Century Center of Excellence Program “New Information Technologies for Building a Networked Symbiotic Environment” and Grant-in-Aid for Scientific Research (A)(17200006) and (18049050) of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

## References

1. Acharya, S., Alonso, R., Franklin, M., and Zdonik, S.: Broadcast disks: data management for asymmetric communication environments. Proc. ACM SIGMOD'95, pp. 199–210, May 1995.
2. Acharya, S., Franklin, M., and Zdonik, S.: Balancing push and pull for data broadcast. Proc. ACM SIGMOD'97, pp. 183–194, May 1997.
3. Aksoy, D., Franklin, M., and Zdonik, S.: Data staging for on-demand broadcast. Proc. VLDB'01, pp. 571–580, Sept. 2001.
4. Kashita, M., Terada, T., Hara, T., Tsukamoto, M., and Nishio, S.: A collaborative query processing method for a database broadcasting system. Proc. CIIT'02, pp. 60–66, Nov. 2002.
5. Kitajima, S., Cai, J., Terada, T., Hara, T., and Nishio, S.: A query processing mechanism based on the broadcast queue for broadcast database systems. Proc. ISWPC'06, pp. 450–455, Jan. 2006.
6. Mahesri, A., and Vardhan, V.: Power consumption breakdown on modern laptop. Proc. International Workshop on Power-Aware Computing Systems (PACS'04), Dec. 2004.

# Mining Vague Association Rules

An Lu, Yiping Ke, James Cheng, and Wilfred Ng

Department of Computer Science and Engineering  
The Hong Kong University of Science and Technology  
Hong Kong, China  
`{anlu, keyiping, csjames, wilfred}@cse.ust.hk`

**Abstract.** In many online shopping applications, traditional Association Rule (AR) mining has limitations as it only deals with the items that are sold but ignores the items that are *almost sold*. For example, those items that are put into the basket but not checked out. We say that those *almost sold* items carry *hesitation information* since customers are hesitating to buy them. The hesitation information of items is valuable knowledge for the design of good selling strategies. We apply vague set theory in the context of AR mining as to incorporate the hesitation information into the ARs. We define the concepts of attractiveness and hesitation of an item, which represent the overall information of a customer's intent on an item. Based on these two concepts, we propose the notion of Vague Association Rules (VARs) and devise an efficient algorithm to mine the VARs. Our experiments show that our algorithm is efficient and the VARs capture more specific and richer information than traditional ARs.

## 1 Introduction

*Association Rule (AR)* mining [1] is one of the most important data mining tasks. Traditional AR mining has been extensively studied for over a decade; however, in recent years, the emergence of many new application domains, such as the Web, has led to many possibilities and challenges of studying new forms of ARs.

Consider the classical market basket case, in which AR mining is conducted on transactions that consist of items bought by customers. However, there are also many items that are not bought but customers may have considered buying them. We call such information on a customer's consideration to buy an item the *hesitation* information of the item, since the customer is hesitating whether to buy it. The hesitation information of an item is useful knowledge for boosting the sales of the item. However, such information is not considered in traditional AR mining due to the difficulty to collect hesitation information in the past. Nevertheless, with the advance in Web technology, it is now much easier to obtain the hesitation information of the items. Consider an online shopping scenario, such as “Amazon.com”, it is possible to collect huge amount of data from the Web log that can be considered as hesitation information. For example, in the online shopping scenario: (1) the items that customers put into their online shopping carts but were not checked out eventually; (2) the items that are in customers' favorite lists to buy next time; (3) the items that are in customers' wishing lists but not yet available in the store; and so on. The hesitation information can then be used to design and implement selling strategies that can potentially turn those “under consideration” items into “well sold” items.

We apply the *vague set theory* [2] as a basis to model the hesitation information of the items. Vague set theory addresses the drawback of a single membership value in *fuzzy set theory* [3] by using interval-based membership that captures three types of evidence with respect to an object in a universe of discourse: *support*, *against* and *hesitation*. Thus, we can naturally model the hesitation information of an item in the mining context as the evidence of hesitation with respect to the item. The information of the “sold” items and the “not sold” items (without any hesitation information) in the traditional setting of AR mining correspond to the evidence of support and against with respect to the item.

To study the relationship between the support evidence and the hesitation evidence with respect to an item, we propose the concepts of *attractiveness* and *hesitation* of an item, which are based on the *median membership* and the *imprecision membership* [4][5] that are derived from the vague membership in vague sets. An item with high attractiveness means that the item is well sold and has a high possibility to be sold again next time. An item with high hesitation means that customers are always hesitating to buy the item due to some reason (e.g., the customer is waiting for price reduction) but has a high possibility to buy it next time, if the reason of giving up the item is identified and resolved (e.g., some promotion on the item is provided).

Using the notions of attractiveness and hesitation of items, we model a database with hesitation information as an *AH*-pair database that consists of *AH*-pair transactions, where *A* stands for attractiveness and *H* stands for hesitation. Based on the *AH*-pair database, we then propose the notion of *Vague Association Rules* (VARs), which capture four types of relationships between two sets of items: the implication of the attractiveness/hesitation of one set of items on the attractiveness/hesitation of the other set of items. To evaluate the quality of the different types of VARs, four types of support and confidence are defined. We also investigate the properties of the support and confidence of VARs, which can be used to speed up the mining process. Based on these properties, an efficient algorithm is then designed to mine the VARs.

Our experiments on both real and synthetic datasets verify that our algorithm to mine the VARs is efficient. Compared with the traditional ARs mined from transactional databases, the VARs mined from the *AH*-pair databases, which are modelled from transactional databases by taking into account the hesitation information of items, are more specific and are able to capture richer information. More importantly, we find that, by aggregating more transactions into an *AH*-pair transaction, our algorithm is significantly more efficient while still obtaining almost the same set of VARs.

**Organization.** This paper is organized as follows. Section 2 presents the VARs and defines related concepts. Section 3 discusses the algorithm to mine the VARs. Section 4 reports the experimental results and Section 5 offers the conclusions.

## 2 Vague Association Rules

In this section, we define the notion of *Vague Association Rules* (VARs) and four types of support and confidence used to evaluate the quality of the VARs. We then present some properties of VARs that can be used to speed up the process of mining VARs.

Given the transactions of the customers, we then aggregate the transactions to obtain the *intent* of each item. Based on the intent of an item, we next define the *attractiveness* and *hesitation* of it.

**Definition 1. (Intent, Attractiveness and Hesitation, AH-Pair Transactions)** *The intent of an item  $x$ , denoted as  $\text{intent}(x)$ , is a vague value  $[\alpha(x), 1 - \beta(x)]$ . The attractiveness of  $x$ , denoted as  $M_A(x)$ , is defined as the median membership of  $x$ , i.e.,  $M_A(x) = (\alpha(x) + (1 - \beta(x))) / 2$ . The hesitation of  $x$ , denoted as  $M_H(x)$ , is defined as the imprecision membership of  $x$ , i.e.,  $M_H(x) = ((1 - \beta(x)) - \alpha(x))$ . The pair  $\langle M_A(x), M_H(x) \rangle$  is called the AH-pair of  $x$ . An AH-pair transaction  $T$  is a tuple  $\langle v_1, v_2, \dots, v_m \rangle$  on an itemset  $I_T = \{x_1, x_2, \dots, x_m\}$ , where  $I_T \subseteq I$  and  $v_j = \langle M_A(x_j), M_H(x_j) \rangle$  is an AH-pair of the item  $x_j$ , for  $1 \leq j \leq m$ . An AH-pair database is a sequence of AH-pair transactions.*

We now present the notion of VARs and define the support and confidence of a VAR.

**Definition 2. (Vague Association Rule)** A Vague Association Rule (VAR),  $r = (X \Rightarrow Y)$ , is an association rule obtained from an AH-pair database.

Based on the attractiveness and hesitation of an item, we define four different types of support and confidence of a VAR depending on what kind of knowledge we want to acquire. For clarity, we use  $A$  to denote *Attractiveness* and  $H$  to denote *Hesitation*.

**Definition 3. (Support)** Given an AH-pair database,  $D$ , we define four types of support for an itemset  $Z$  or a VAR  $X \Rightarrow Y$ , where  $X \cup Y = Z$ , as follows.

1. The  $A$ -support of  $Z$ , denoted as  $\text{Asupp}(Z)$ , is defined as  $\sum_{T \in D} \prod_{z \in Z} M_A(z) / |D|$ .
2. The  $H$ -support of  $Z$ , denoted as  $\text{Hsupp}(Z)$ , is defined as  $\sum_{T \in D} \prod_{z \in Z} M_H(z) / |D|$ .
3. The AH-support of  $Z$ , denoted as  $\text{AHsupp}(Z)$ , is defined as  $\sum_{T \in D} \prod_{x \in X, y \in Y} M_A(x) M_H(y) / |D|$ .
4. The HA-support of  $Z$ , denoted as  $\text{HASupp}(Z)$ , is defined as  $\sum_{T \in D} \prod_{x \in X, y \in Y} M_H(x) M_A(y) / |D|$ .

$Z$  is an  $A$  (or  $H$  or  $AH$  or  $HA$ ) FI if the  $A$ - (or  $H$ - or  $AH$ - or  $HA$ -) support of  $Z$  is no less than the (respective  $A$  or  $H$  or  $AH$  or  $HA$ ) minimum support threshold  $\sigma$ .

**Definition 4. (Confidence)** Given an AH-pair database,  $D$ , we define the confidence of a VAR,  $r = (X \Rightarrow Y)$ , where  $X \cup Y = Z$ , as follows.

1. If both  $X$  and  $Y$  are  $A$  FIs, then the confidence of  $r$ , called the  $A$ -confidence of  $r$  and denoted as  $\text{Aconf}(r)$ , is defined as  $\frac{\text{Asupp}(Z)}{\text{Asupp}(X)}$ .
2. If both  $X$  and  $Y$  are  $H$  FIs, then the confidence of  $r$ , called the  $H$ -confidence of  $r$  and denoted as  $\text{Hconf}(r)$ , is defined as  $\frac{\text{Hsupp}(Z)}{\text{Hsupp}(X)}$ .
3. If  $X$  is an  $A$  FI and  $Y$  is an  $H$  FI, then the confidence of  $r$ , called the AH-confidence of  $r$  and denoted as  $\text{AHconf}(r)$ , is defined as  $\frac{\text{AHsupp}(Z)}{\text{Asupp}(X)}$ .

4. If  $X$  is an  $H$  FI and  $Y$  is an  $A$  FI, then the confidence of  $r$ , called the HA-confidence of  $r$  and denoted as  $\text{HAconf}(r)$ , is defined as  $\frac{\text{HAsupp}(Z)}{\text{Hsupp}(X)}$ .

**Problem Description.** Given an AH-pair database  $D$ ,  $\sigma$  and  $c$ , the problem of VAR mining is to find all VARs  $r$  such that  $\text{supp}(r) \geq \sigma$  and  $\text{conf}(r) \geq c$ , where  $\text{supp}$  and  $\text{conf}$  are one of the  $A$ -,  $H$ -,  $AH$ -, and  $HA$ - support and confidence.

Note that the thresholds  $\sigma$  and  $c$  can be different for different types of VARs. Hereafter, we just set them to be the same for different types of VARs, and this can be easily generalized to the case of different thresholds.

We give some properties of VARs which can be used to design an efficient algorithm for mining VARs. The following proposition states that the support defined for an itemset in an AH-pair database has the anti-monotone property.

**Proposition 1.** *The following statements are true.*

1. If  $X \subseteq X'$ , then  $\text{Asupp}(X') \leq \text{Asupp}(X)$  and  $\text{Hsupp}(X') \leq \text{Hsupp}(X)$ .
2. Given an item  $x$ ,  $\frac{M_H(x)}{2} \leq M_A(x) \leq 1 - \frac{M_H(x)}{2}$ .
3. Given a VAR,  $r = (X \Rightarrow Y)$ , where  $|X| = m$  and  $|Y| = n$ , we have  $(\frac{1}{2})^m \text{Hsupp}(r) \leq \text{AHsupp}(r) \leq 2^n \text{Asupp}(r)$ ;  $(\frac{1}{2})^n \text{Hsupp}(r) \leq \text{HAsupp}(r) \leq 2^m \text{Asupp}(r)$ ;  $\text{AHconf}(r) \leq 2^n \text{Aconf}(r)$ ;  $(\frac{1}{2})^n \text{Hconf}(r) \leq \text{HAconf}(r)$ .

### 3 Mining Vague Association Rules

In this section, we present an algorithm to mine the VARs. We mine the set of all  $A$ ,  $H$ ,  $AH$  and  $HA$  FIs from the input AH-pair database, and then generate the VARs from FIs.

Let  $A_i$  and  $H_i$  be the set of  $A$  FIs and  $H$  FIs containing  $i$  items, respectively. Let  $A_i H_j$  be the set of  $AH$  FIs containing  $i$  items with  $A$  values and  $j$  items with  $H$  values. Note that  $A_i H_j$  is equivalent to  $H_j A_i$ . Let  $C_S$  be the set of *candidate FIs*, from which the set of FIs  $S$  is to be generated, where  $S$  is  $A_i$ ,  $H_i$ , or  $A_i H_j$ .

---

#### Algorithm 1. MineVFI( $D, \sigma$ )

---

1. Mine  $A_1$  and  $H_1$  from  $D$ ;
  2. Generate  $C_{A_2}$  from  $A_1$ ,  $C_{A_1 H_1}$  from  $A_1$  and  $H_1$ , and  $C_{H_2}$  from  $H_1$ ;
  3. Verify the candidate FIs in  $C_{A_2}$ ,  $C_{A_1 H_1}$  and  $C_{H_2}$  to give  $A_2$ ,  $A_1 H_1$  and  $H_2$ , respectively;
  4. **for each**  $k = 3, 4, \dots$ , where  $k = i + j$ , **do**
  5.     Generate  $C_{A_k}$  from  $A_{i-1}$  and  $C_{H_k}$  from  $H_{i-1}$ , for  $i = k$ ;
  6.     Generate  $C_{A_i H_j}$  from  $A_{i-1} H_j$ , for  $2 \leq i < k$ , and from  $A_1 H_{j-1}$ , for  $i = 1$ ;
  7.     Verify the candidate FIs in  $C_{A_k}$ ,  $C_{H_k}$ , and  $C_{A_i H_j}$  to give  $A_k$ ,  $H_k$ , and  $A_i H_j$ ;
  8. **return** all  $A_i$ ,  $H_j$ , and  $A_i H_j$  mined;
- 

The algorithm to compute the FIs is shown in Algorithm 1. We first mine the set of frequent items  $A_1$  and  $H_1$  from the input AH-pair database  $D$ . Next, we generate the candidate FIs that consists of two items (Line 2) and compute the FIs from the candidate FIs (Line 3). Then, we use the FIs containing  $(k-1)$  items to generate the candidate FIs containing  $k$  items, for  $k \geq 3$ , which is described as follows.

For each pair of FIs,  $x_1 \cdots x_{k-2}y$  and  $x_1 \cdots x_{k-2}z$  in  $A_{k-1}$  or  $H_{k-1}$ , we generate the itemset  $x_1 \cdots x_{k-2}yz$  into  $C_{A_k}$  or  $C_{H_k}$ . For each pair of FIs,  $x_1 \cdots x_{i-2}uy_1 \cdots y_j$  and  $x_1 \cdots x_{i-2}vy_1 \cdots y_j$  in  $A_{i-1}H_j$ , or  $x_1y_1 \cdots y_{j-2}u$  and  $x_1y_1 \cdots y_{j-2}v$  in  $A_1H_{j-1}$ , we generate the itemset  $x_1 \cdots x_{i-2}uvy_1 \cdots y_j$  or  $x_1y_1 \cdots y_{j-2}uv$  into  $C_{A_iH_j}$ .

After generating the candidate FIs, we obtain the FIs as follows. For each  $Z \in C_{A_k}$  (or  $Z \in C_{H_k}$ ), if  $\exists X \subset Z$ , where  $X$  contains  $(k-1)$  items,  $X \notin A_{k-1}$  (or  $X \notin H_{k-1}$ ), then we remove  $Z$  from  $C_{A_k}$  (or  $C_{H_k}$ ). For each  $Z = x_1 \cdots x_iy_1 \cdots y_j \in C_{A_iH_j}$ , if  $\exists i'$ , where  $1 \leq i' \leq i$ ,  $(Z - \{x_{i'}\}) \notin A_{i-1}H_j$ ; or  $\exists j'$ , where  $1 \leq j' \leq j$ ,  $(Z - \{y_{j'}\}) \notin A_iH_{j-1}$ , then we remove  $Z$  from  $C_{A_iH_j}$ . Here, the *anti-monotone property* [1] of support is applied to prune  $Z$  if any of  $Z$ 's subsets is not an FI. After that, the support of the candidate FIs is computed and only those with support at least  $\sigma$  are retained as FIs. Finally, the algorithm terminates when no candidate FIs are generated and returns all FIs.

After mining the set of all FIs, we generate the VARs from the FIs. There are four types of VARs. First, for each  $A$  or  $H$  FI  $Z$ , we can generate the VARs  $X \Rightarrow Y, \forall X, Y$  where  $X \cup Y = Z$ , using the classical AR generation algorithm [1]. Then, for each  $AH$  (or  $HA$ ) FI  $Z = (X \cup Y)$ , where  $X$  is an  $A$  FI and  $Y$  is an  $H$  FI, we generate two VARs  $X \Rightarrow Y$  and  $Y \Rightarrow X$ . The confidence of the VARs can be computed by Definition 4.

## 4 Experiments

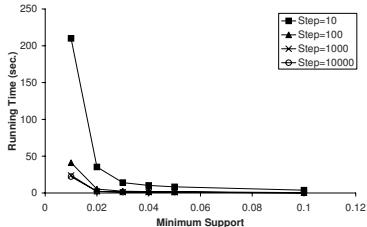
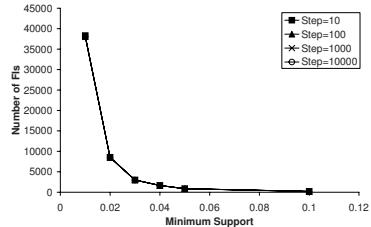
In this section, we use both real and synthetic datasets to evaluate the efficiency of the VAR mining algorithm and the usefulness of the VARs. All experiments are conducted on a Linux machine with an Intel Pentium IV 3.2GHz CPU and 1GB RAM.

### 4.1 Experiments on Real Datasets

For the first set of experiments, we use the Web log data from IRCCache [6], which is the NLANR Web Caching project. Then we can classify Web pages into three categories: *target*, *non-target*, and *transition* according to the time spent on the Web page, the position of the Web page in the browsing trail and the number of visits to the Web page. The three categories correspond to the three status of items, i.e., 1, 0 and  $h$ .

Since the Web log data contain a huge number of different Web sites, we only report the result on the Web log of a single Web site ([www.google.com](http://www.google.com)) from all nine IRCCache servers on a single day (Aug. 29, 2006). When  $\sigma=0.001$  and  $c=0.9$ , we obtain one VAR:  $http://gmail.google.com/, http://gmail.google.com/mail/ \Rightarrow http://mail.google.com/mail/$ , with HA-support of 0.003 and HA-confidence of 0.99. This VAR shows that  $http://gmail.google.com/$  and  $http://gmail.google.com/mail/$  always play the role of transition pages to the target page  $http://mail.google.com/mail/$ . As a possible application, we can add a direct link from the transition pages ( $http://gmail.google.com/$  or  $http://gmail.google.com/mail/$ ) to the target page ( $http://mail.google.com/mail/$ ) to facilitate the user traversal of the Web site. Actually, by typing either the URL of the two transition pages in a Web browser, it is redirected to the URL of the target page, where the redirect mechanism serves as a special kind of direct link.

In order to compare with the traditional ARs, we also test on the database that contains all the trails without distinguishing the Web pages. At  $\sigma=0.0008$  and  $c=1$ , 70 ARs

**Fig. 1.** Running Time**Fig. 2.** Number of FIs

are returned. Among them, 59 ARs (84%) contain the entrance page ([www.google.com](http://www.google.com)), which is not that interesting. Among the remaining ARs, the following rule is found:  $http://mail.google.com/ \Rightarrow http://mail.google.com/mail/ \Rightarrow http://mail.google.com/mail/$  with support 0.001 and confidence 1, which is similar to the VAR we find. This result shows the effectiveness of mining VARs, since the traditional AR mining approach returns many ARs but it is difficult for the user to tell which ARs are more important for practical uses, while mining VARs can find more specific rules directly.

#### 4.2 Experiments on Synthetic Datasets

We test on the synthetic datasets to evaluate the efficiency and the scalability of our algorithm. We modify the IBM synthetic data generator [7] by adding ‘‘hesitation’’ items. The ID and the number of ‘‘hesitation’’ items in each transaction are generated according to the same distributions as those for the original items. We generate a dataset with 100000 transactions and 100 items. We use a parameter *Step* to represent the number of transactions which are aggregated to give an *AH*-pair transaction.

We first test the algorithm under different values of  $\sigma$ . Fig. 1 and Fig. 2 report the running time and the number of FIs. From Fig. 1 the running time increases with the decrease in the value of  $\sigma$  due to the larger number of FIs generated. We also find that, for the same value of  $\sigma$ , the running time decreases significantly with the increase in the value of *Step*. This is because we aggregate more transactions to a single *AH*-pair transaction and hence the number of *AH*-pair transactions is smaller in the database. However, Fig. 2 shows that the number of FIs for the different *Step* values varies only slightly (note that all the four lines are coincided into one line in Fig. 2). We further check the FIs obtained for the different *Step* values and find that they are indeed similar. This result shows that we can actually aggregate more transactions to give the *AH*-pair transactions so that we can improve the efficiency of the mining operation but still obtain the same set of FIs and hence the VARs.

### 5 Conclusions

We apply the vague set theory to address a limitation in traditional AR mining problem, that is, the hesitation information of items is not considered. We propose the notion of

VARs that incorporates the hesitation information of items into ARs. We also define different types of support and confidence for VARs in order to evaluate the quality of the VARs for different purposes. An efficient algorithm is proposed to mine the VARs, while the effectiveness of VARs is also confirmed by the experiments on real datasets.

## References

1. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In Buneman, P., Jajodia, S., eds.: SIGMOD Conference, ACM Press (1993) 207–216
2. Gau, W.L., Danied, J.B.: Vague sets. IEEE Transactions on Systems, Man, and Cybernetics **23** (1993) 610–614
3. Zadeh, L.A.: Fuzzy sets. Information and Control **8** (1965) 338–353
4. Lu, A., Ng, W.: Managing merged data by vague functional dependencies. In Atzeni, P., Chu, W.W., Lu, H., Zhou, S., Ling, T.W., eds.: ER. Volume 3288 of Lecture Notes in Computer Science., Springer (2004) 259–272
5. Lu, A., Ng, W.: Vague sets or intuitionistic fuzzy sets for handling vague data: Which one is better? In Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, O., eds.: ER. Volume 3716 of Lecture Notes in Computer Science., Springer (2005) 401–416
6. NLANR: (<http://www.ircache.net/>)
7. IBM Quest Data Mining Project. The Quest retail transaction data generator. <http://www.almaden.ibm.com/software/quest/> (1996)

# An Optimized Process Neural Network Model\*

Guojie Song<sup>1,3</sup>, Dongqing Yang<sup>1</sup>, Yunfeng Liu<sup>2</sup>, Bin Cui<sup>1</sup>, Ling Wu<sup>1</sup>,  
and Kunqing Xie<sup>3</sup>

<sup>1</sup> School of Electronic Engineering and Computer Science, Peking University, Beijing, China  
`gjsong@pku.edu.cn, cuibin@pku.edu.cn, dqyang@pku.edu.cn`

<sup>2</sup> Computer Center of Peking University, Beijing

<sup>3</sup> National Laboratory on Machine Perception, Peking University, Beijing  
`kunqing@cis.pku.edu.cn`

**Abstract.** In this paper, we proposed an optimized process neural network based on fourier orthogonal base function, which can deal with both static value and time-varied continuous value simultaneously. To further improve its performance, we optimize the network topological structure, which adopts fourier expansion based preprocessing. Experiments based on the real datasets show that our proposed churn prediction method has better maneuverability and performance. Most important of all, our method has been used in real applications in China Mobile which is the major telecommunication company of the world.

## 1 Introduction

Many techniques have emerged for the purpose of classification, such as artificial neural networks, SVM and classification trees [12][5][6]. Unfortunately, inputs of all these methods are static values. However, real applications also include many time-varied continuous values. Such time continuous values are always being summarized firstly by using techniques, such as *sum*, *average* etc, and then taken as input in static value by existing methods. However, many useful information for churn prediction are contained in such time-varied continuous values, but which will be removed with the execution of above preprocessing unfortunately.

To solve this problem, He and Liang proposed artificial process neuron model [3] in 2000. From a point view of architecture, process neuron is similar to conventional artificial neuron, which can be considered as a general form of traditional neuron model. The major difference is that the input, the output and the corresponding connection weight of process neuron are not static values but time-varied functions, and capable of imitating time-varied continuous value perfectly. Process neural network (PNN) is composed of densely interconnected process neurons. A particularly important element of designing process neuron is the choice of base function, which can influence its performance greatly. In general, the characteristics of PNN is its high prediction accuracy and express ability. But the efficiency of PNN is always concerned by users.

In this paper, an optimized process neural network, named MPNN, has been proposed, which can deal with both traditional static data and the time-varied continuous

---

\* This work is supported by the National Natural Science Foundation of China under Grant No. 60473051 and No.60642004 and IBM and HP Joint Research Project.

data simultaneously. Fourier orthogonal base function has been chosen as the base function of process neuron to expand each time varied continuous series efficiently. To avoid repeating computation of fourier base function expansion and time aggregation operation, an optimized MPNN has been proposed, which use the preprocessing technique based on fourier transform to simplify the structure of MPNN. The effectiveness and efficiency of MPNN have also been proved by our extensively experiments based on the real dataset. The system with the proposed churn prediction model has been implemented, and used in China Mobile Communications.

The remaining of the paper is organized as follows. In Section 2, we presents the proposed MPNN model. A performance study of the proposed method is demonstrated in Section 3, and finally we conclude our study in Section 4.

## 2 MPNN: A Mixed PNN Model

### 2.1 Topological Structure of MPNN

The MPNN proposed is composed of four layers: input layer, two hidden layers and one output layer. Input layer is composed of  $n + \sum_{i=1}^m A_i$  units, which includes both continuous time-varied data, such as billing data, and discrete data, such as customer gender etc. Thus, the input of MPNN include  $n$  input node dealing with continuous time series data and  $\sum_{i=1}^m A_i$  nodes for discrete static input data, where each input node  $d_{i,A_i}$  corresponds to one value of property  $A_i$ , with 1 if  $\text{value}(A_i) = d_{i,A_i}$  else with 0. The first hidden layer is composed of  $n$  process neurons and  $m$  traditional neurons. Process neurons deal with  $n$  time continuous input data and traditional neuron only accept  $\sum_{i=1}^m A_i$  discrete static data from the input layer. The second hidden layer is composed of  $p$  traditional neurons and the last layer is output layer. To reduce the complexity, we will only consider the case of one output unit.

### 2.2 Relationship Between Input and Output

**Input:** The inputs of the first layer can be expressed as two parts:  $X(t)$  for time varied continuous time series and  $D$  for discrete static input, denoted as  $X(t) = (x_1(t), x_2(t), \dots, x_n(t))D = (d_{1,1}, \dots, d_{1,A_1}, \dots, d_{m,1}, \dots, d_{m,A_m})$

**The outputs of the first hidden layer:** We first consider the computation of  $j$ -th process neuron connected with continuous input  $X(t)$ .  $y_j^{(c,1)} = f_p(\sum_{i=1}^n \int_0^T w_{ci,j}(t)x_i(t)dt)$ , where  $w_{ci,j}(t)$  is the link weight function between  $j$ -th process neuron in the first hidden layer and the  $i$ -th unit of  $X(t)$  in the input layer ( $i, j \in [1, n]$ ).

If we take fourier base function  $s_i(t)$ ,  $i \in [1, L]$ , as process base function to expand process weight function and input  $x_i(t)$ , formula 2.2 will be

$$\begin{aligned} y_j^{(c,1)} &= f_p\left(\sum_{i=1}^n \int_0^T \left(\sum_{q=1, z=1}^{q=L, z=L} c_{iz}^{(z)} w_{ciq}^{(q)} s_q(t) s_z(t)\right) dt\right) \\ &= f_p\left(\sum_{i=1}^n \sum_{q=1, z=1}^{q=L, z=L} c_{iz}^{(z)} w_{ciq}^{(q)} \int_0^T s_q(t) s_z(t) dt\right) \end{aligned} \quad (1)$$

Thus, the outputs in the first hidden layer can be expressed as

$$y_j^{(c,1)} = \sum_{i=1}^n \sum_{q=1,z=1}^{q=L,z=L} c_{iz}^{(z)} w c_{iq}^{(q)} \quad (2)$$

where  $f_p$  is the activation function of the process neurons in the first hidden layer.

For traditional neurons in the first layer, its outputs can be expressed as  $y_k^{(d,1)} = f_d(\sum_{h=1}^{A_k} d_{kh} w d_{hk})$ , where  $w d_{hk}$  is the link weight between  $k$ -th neuron in the first hidden layer and the  $h$ -th value of property  $d_k$  in the discrete input layer ( $k \in [1, m]$ ,  $h \in [1, A_k]$ ). Thus, the outputs of the first hidden layer is  $out_r^{(1)} = y_j^{(c,1)} + y_h^{(d,1)}$ , where  $r \in [1, m+n]$ .

**The outputs of the second hidden layer:** Based on the outputs of the first hidden layer, the outputs of the second hidden layer can be expressed as  $y_l^{(2)} = f_g(\sum_{r=1}^{m+n} out_r^{(1)} v_{rl})$ , where  $v_{rl}$  is the link weight between the  $l$ -th neuron in the second hidden layer and the  $r$ -th neuron in the first hidden layer.  $f_g$  is the activation function of the neuron in the second hidden layer.

**Output:** The output function of the MPNN can be expressed as  $y(t) = f_o(\sum_{l=1}^p y_l^{(2)} u_l)$ , where  $u_l$  is the link weight between the  $l$ -th neuron in the second hidden layer and the output node.  $f_o$  is the activation function of the output node.

### 2.3 Learning Algorithm

Assume that we have  $K$  learning sample functions:

$$\begin{bmatrix} x_{11}(t), x_{12}(t), \dots, x_{1n}(t), y_1(t) \\ x_{21}(t), x_{22}(t), \dots, x_{2n}(t), y_2(t) \\ \vdots \quad \vdots \\ x_{K1}(t), x_{K2}(t), \dots, x_{Kn}(t), y_K(t) \end{bmatrix}$$

where the first suffix  $i$  in  $x_{ij}(t)$  denotes the serial number of learning sample, and the second suffix  $j$  denotes the serial number of component in input function vector.  $y_k(t)$  is the expected output function for input  $x_{k1}(t), x_{k2}(t), \dots, x_{kn}(t)$ , ( $k \in [1, K]$ ).

Suppose that  $\tilde{y}(t)$  is the desired output function, and  $y(t)$  is the corresponding actual output function of the MPNN, then the mean square error of the MPNN output can be written as

$$\begin{aligned} E &= \frac{1}{2} \sum_{b=1}^K (y_b(t) - \tilde{y}_b(t))^2 = \frac{1}{2} \sum_{b=1}^K \left[ f_o \left( \sum_{l=1}^p y_{lb}^{(2)} u_l \right) - \tilde{y}_b(t) \right]^2 \\ &= \frac{1}{2} \sum_{b=1}^K \left\{ f_o \left[ \sum_{l=1}^p f_g \left( \sum_{r=1}^{m+n} \left( f_p \left( \sum_{i=1}^n \sum_{q=1,z=1}^{q=L,z=L} c_{izb}^{(z)} w c_{iq}^{(q)} \right) + f_d \left( \sum_{h=1}^{A_k} d_{khb} w d_{hk} \right) \right) v_{rl} \right) u_l \right] - \tilde{y}_b(t) \right\}^2 \end{aligned}$$

For analysis convenience,  $Z_b$ ,  $Q_b$ ,  $P_b$  and  $H_b$  are defined respectively as

$$\begin{aligned} H_b &= (\sum_{h=1}^{A_k} d_{khb} w d_{hk}), \quad P_b = \sum_{i=1}^n \sum_{q=1,z=1}^{q=L,z=L} (c_{izb}^{(z)} w c_{iq}^{(q)}) \\ Q_b &= \sum_{r=1}^{m+n} (f_d(H_b) + f_p(P_b)) v_{rl}, \quad Z_b = \sum_{l=1}^p Z_b u_l \end{aligned}$$

According to the gradient descent method, the learning rules are defined as follows  
 $wc_{iq}^{(q)} = wc_{iq}^{(q)} + \alpha \Delta, wc_{iq}^{(q)} wd_{hk} = wd_{hk} + \beta \Delta wd_{hk}, v_{rl} = v_{rl} + \gamma \Delta v_{rl}, u_l = u_l + \eta \Delta u_l$ , where  $\alpha, \beta, \gamma, \eta$  are the learning rate, and  $i \in [1, n], k \in [1, m], h \in [1, A_k], l \in [1, p], q, r \in [1, L]$ .

$\Delta wc_{ik}^{(k)}, \Delta wd_{jh}, \Delta v_{ik}$ , and  $\Delta u_k$  can be calculated as follows

$$\Delta wc_{iq}^{(q)} = -\frac{\partial E}{\partial wc_{iq}^{(q)}} = -\Omega u_l f'_g(Q_b) v_{rl} f'_p(P_b) c_{izb}^{(z)},$$

$$\Delta wd_{hk} = -\frac{\partial E}{\partial wd_{hk}} = -\Omega u_l f'_g(Q_b) v_{rl} f'_d(H_b) d_{kh}^{(z)},$$

$$\Delta v_{rl} = -\frac{\partial E}{\partial v_{rl}} = -\Omega u_l f'_g(Q_b), \Delta u_l = -\frac{\partial E}{\partial u_l} = -\Omega, \text{ where } \Omega = \sum_{b=1}^K (f_o(Z_b) - \tilde{y}_b(t)) f'_o(Z_b).$$

In this paper, all activation functions have been substituted by Sigmoid function, i.e.  $f_o(u) = f_g(u) = f_p(u) = f_d(u) = \frac{1}{1+e^u}$ , with  $f'_*(u) = f_*(u)(1 - f_*(u))$ .

## 2.4 Topological Structure

Based on formula 2 if we take fourier base function as the base function of process neuron in MPNN, each input time varies function  $x_i(t)$  can be expanded with a set of fourier coefficient  $c_i$  and corresponding weight  $w_i$ . If we take  $x_i(t)$  as input of process neuron in MPNN during each time iteration of training process, it should be expanded one time by using DFT accompanied with one time aggregation operation by using formula  $\int_0^T s_q(t) s_z(t) dt$ . In fact, such costs of the fourier expansion can be avoided if each input time series  $x_i(t), i \in [1, n]$ , has been preprocessed before it is input into MPNN. Because  $c_i$  is a constant and has no relationship with the training process, so taking fourier coefficient  $c_i$  as input will not influence the final results. By using such preprocessing strategy, the process neuron in MPNN have become a traditional neuron by losing its time aggregation ability.

## 3 Data and Experimental Results

### 3.1 Characteristics of Input Data

We get the real data from the China Mobile Communication Company for Churn prediction. We sample the dataset from Jan. 2004 to April 2004. After filtering the data with missing values, we select 220 thousands samples. 2000 samples have been selected randomly for training data set and 10000 samples for testing data set. The ratio of churner is 20%. The description of the variables for this research is presented as follows.

**Time varied continuous data:** It includes three kinds of usage series data: *call time*, *the number of messages* and *the number of different telephones communicated with him/her*. Each data element in series is accumulated in term of day, and 91 element spanning three months have been generated in each series.

**Traditional static discrete data:** Two discrete variables have been selected, which are customer's *gender*(male:0, female:1) and *age* (being discreted into five segments, (0-20), (20-30), (30, 40), (40, 60) and (60-100) in advance).

**Mark of churn:** According to the definition of churn management strategies, different in each province, the mark of churn for each customer can be defined with whether he is churn in (churn:1, nonchurn:0), that is in June 2004.

### 3.2 Experimental Results

All experiments are conducted on a PC with Pentium IV 1.4G CPU and 512MB main memory, running Windows XP operation system. The MPNN code was written in C++. Some existing methods, e.g. PNN, ANN [6] and Decision Tree(C4.5)[5] have been compared. The neural networks used in our experiments are multilayer perceptrons with a single hidden layer which contains 10 nodes and they were trained by the back propagation algorithm with the learning rate 0.3 and the momentum term 0.7. The triangle function has been chosen as base function of PNN, and the input is three kinds of time continuous data. Continuous data have been averaged before input ANN and C4.5. We compared their performance about predicting accuracy, lift value and execution time. The model predicts correctly with churn if the predicted user's churn probability is bigger than (or equal to) 0.5 or the user doesn't churn with the probability less than 0.5. Otherwise the Model predicts wrong.

**Precision measurement:** In this section, two basic measures, *precision* and *recall*, have been introduced to evaluate our prediction methods. The experimental results can be seen in Table I. It can be observed that our method MPNN has the best performance in detecting the churn with precision 87.5% and recall 81.5%. C4.5 and ANN is less than PNN no matter for precision or recall.

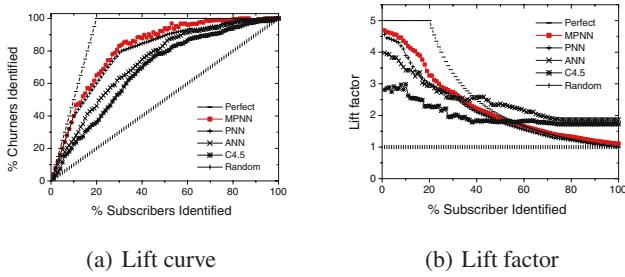
**Table 1.** Comparison of different Algorithms

| Algorithm | ANN   | C4.5   | PNN   | MPNN  |
|-----------|-------|--------|-------|-------|
| Precision | 78%   | 71.2 % | 83.3% | 87.5% |
| Recall    | 73.3% | 69.2 % | 78.2% | 81.5% |

**Lift value measurement:** We applied MPNN to the training dataset to predict the *churn* or *no churn* of the subscribers in the testing dataset. In the telecommunications industry, the *churn* and *no churn* prediction is usually expressed as a lift curve. The lift curve plots the fraction of all churners having churn probability above the threshold against the fraction of all subscribers having churn probability above the threshold.

The lift curves are shown in Figure II(a). As described in figure, when compared with C4.5 and ANN, MPNN identified more churners than them under the same fraction of subscribers. It is important to note that PNN also identified more churners than C4.5 and ANN. When compared with PNN, MPNN identified more churners than PNN did.

To better compare the performance of these models, let us consider the lift factor, which is defined as the ratio of the fraction of churners identified and the fraction of subscribers contacted. It is important to note that the lift factor for the random churn predictor is 1. Owing to the limited number of staff in the carrier's customer services center, it can only contact 5% of all subscribers. The lift factors for these models were contacted under different fraction of subscribers are shown in Figure II(b). Again, MPNN obtained

**Fig. 1.** Lift Measurement**Table 2.** Comparison of Execution Time(sec)

| SampleData | ANN     | C4.5  | PNN    | MPNN    |
|------------|---------|-------|--------|---------|
| 2000       | 21.23   | 11.8  | 43.3   | 24.48   |
| 4000       | 45.24   | 25.2  | 82.9   | 47.078  |
| 6000       | 76.2    | 37.7  | 123.47 | 77.094  |
| 8000       | 93.134  | 59.32 | 164.3  | 99.094  |
| 10000      | 101.725 | 79.77 | 214.3  | 108.812 |

higher lift factors than PNN, which in turn obtained higher lift factors than ANN and C4.5 when the first 20% subscriber are contacted.

**Computation efficiency measurement:** To evaluate their computation efficiencies, Table 2 shows the execution times for MPNN, PNN, C4.5 and ANN under different sample data size.

The experimental results showed that MPNN accomplished the churn prediction task almost the same as ANN, since fourier base function can deal with continuous data in linear time, and make MPNN with the same function as ANN. Of the four approaches, C4.5 required the least execution time to complete since C4.5 used less number of iterations than the rest three models. However, C4.5 is unable to produce churn prediction as accurate as others (as described in Table 1, Figures 1). Thus, our MPNN model not only achieved higher accuracy but also with less execution time.

## 4 Conclusion

In this paper, we proposed an optimized process neural network model, and its performance has been investigated. The result shows that the classification accuracy of MPNN, 87.15%, is better than of ANN, Decision Tree (C4.5).

## References

1. Sun Kim, Kyung-shik Shin, Kyungdo Park: An Application of Support Vector Machines for Customer Churn Analysis: Credit Card Case. ICNC (2) 2005: 636-647.
2. Mozer, M.C., Wolniewicz, R., Grimes, D.B.: Predicting Subscriber's Dissatisfaction and Improving Retention in the Wireless Telecommunications Industry. IEEE Transactions on Neural Networks, Vol. 11, 3 (2000) 690-696.

3. He X.G., Liang J.Z. Some Theoretical Issues on Process Neural Networks. *Engineering Science*, 2 (2000) 40-44.
4. HE X.G. , XU S.H. A feedback process neuron network model and its learning algorithm[J ]. *Acta Autotomatica Sinica* , 2004 , 30 (6) :801 - 806.
5. C. Bishop, *Neural Networks for Pattern Recognition*. New York: Oxford Univ. Press, 1995.
6. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

# Clustering XML Documents Based on Structural Similarity

Guangming Xing<sup>1</sup>, Zhonghang Xia<sup>1</sup>, and Jinhua Guo<sup>2</sup>

<sup>1</sup> Department of Computer Science, Western Kentucky University, Bowling Green,  
KY 42104

[guangming.xing@wku.edu](mailto:guangming.xing@wku.edu), [zhonghang.xia@wku.edu](mailto:zhonghang.xia@wku.edu)

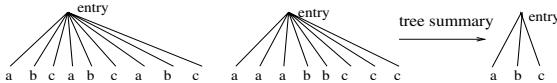
<sup>2</sup> Computer and Information Science Department, University of Michigan - Dearborn,  
Dearborn, MI 48128  
[jinhua@umich.edu](mailto:jinhua@umich.edu)

**Abstract.** In this paper, we present a framework for clustering XML documents based on structural similarity between XML documents. Firstly, the validity of using the edit distance between XML documents and schemata as the structural similarity is presented. Secondly, a novel solution is given for schema extraction. The solution is based on the minimum length description (MLD) principle, and allows tradeoff between the schema simplicity and precision based on the user's specification. Thirdly, clustering XML documents based on the edit distance is discussed. The efficacy and efficiency of our methodology have been tested using both real and synthesized data.

## 1 Motivation and Literature Review

The widely use of XML in different business applications results in large volume of heterogeneous data: XML documents conforming to different schemata. An XML document is defined by the markup tags from a *Document Type Definition (DTD)*, forming a tree structure. Clustering XML documents based on the tree structure is an important problem and is crucial for XML document storage and retrieval [9].

Various methods [8][3] have been proposed and implemented for XML document clustering, and most of them use tree edit distance as a measure of similarity. Tree edit distance [6] is defined as the cost of the sequence of edit operations to transform one tree to another. It offers a very precise measure for document similarity between two documents. However, tree edit distance is not a good measure for structural similarity, as edit distance between two documents can be large (one large document and one small document) while they have very similar structures (conform to the same schema). To overcome this problem, various methods have been proposed. For example, Jagadish [8] proposed a method using *graft* and *prune* to improve the efficiency of computing edit distance and accuracy of clustering. More recently, Dalamagas [3] studied XML document clustering using tree summaries and top-down tree edit distance. Both methods offers very high clustering accuracy when the set of documents



**Fig. 1.** Trees of Different Structure with the Same Structural Summary

conforms to the DTDs whose length of the repeat patterns is 1. However, the performance of these two methods get significantly degraded when the underlying DTDs have more complicated patterns. Although the tree summary method significantly reduces the time complexity for computing the tree edit distance, the structures of the trees may not be preserved by the structural summaries. For example, consider the example in Fig. 1: the two trees on the left side have different structures, but they share the same structural summary based on the methods in [3], which can be illustrated by the tree on the right side in Fig. 1.

Based on the above observations and the fact that the structure of an XML document is defined by a schema, it is natural to study the distance between XML documents and schemata and use it as a similarity measure for document clustering. In this paper, we use the methods presented in [2] to compute the edit distance.

The remainder of this paper is organized as follows. The algorithm to find a schema that can generate a set of XML documents is covered in Section 2. Section 3 covers the use of the edit distance between an XML document and a schema in clustering XML documents. The implementation and experimental studies are presented and discussed in Section 4, and the conclusion remarks are given in Section 5.

## 2 Schema Extraction from XML Documents

Most clustering methods rely on pairwise distances. In order to use the edit distance defined in [2] for document clustering, the schemata of the documents are needed. Therefore, the first task is to extract the underlying schema from XML documents. The problem can be formulated as: Given a collection of XML documents  $\{d_1, d_2, \dots, d_n\}$ , find a schema  $s$ , such that  $d_1, d_2, \dots, d_n$  are document instances that can be generated by schema  $s$ .

The definition of an element in a schema is independent of the definitions of other elements, and only restricts the sequence of sub-elements (the attributes are omitted in this paper) nested within the element. Therefore, the schema extraction can be simplified as inferring a regular expression (right linear grammar or nondeterministic finite automata) from a collection of strings.

Inferring regular expressions from a set of strings has been well studied in [11,10]. One novel contribution by Xtract is the introduction of Minimum Length Description (MLD) to rank candidate expressions. In general, the MLD principle states that the best grammar (schema) to infer from a set of data is the one that minimizes the sum of: the length of the grammar  $L_g$ , and the length of the data  $L_d$  when encoded with the grammar.

In this paper, we use a similar approach as introduced in Xtract: Candidate regular expressions are extracted based on the analysis of the repeat patterns appearing in the input sequences. The candidate expressions are then ranked using MLD principle. We have made the following improvements over Xtract:

1. The frequencies of the children sequences are considered in our system. This feature helps to minimize the negative effects of noise data in clustering.
2. In our system, the relative weight between the definition and description can be dynamically adjusted. The overall goal in our system is to minimize  $L = \lambda L_g + L_d$ . The  $\lambda$  can be used to adjust the precision and generalness of the result.
3. Instead of using a regular expression to determine the cost of encoding, we use the cost of nondeterministic finite automata (NFA) simulation as the cost of encoding. This eliminates the necessity for enumerate multiple sequence partitions to compute the minimum cost for encoding.

It is difficult to represent the encoding of a string with a regular expression. So instead of working on regular expressions, we consider NFA constructed by Thompson's [\[4\]](#) method. The NFA can be represented by encoding the states and its transitions. So  $L_g$  can be represented as  $(S + T) \log S$ , where  $S$  is the number of states and  $T$  is the number of transitions.

To compute  $L_d$ , we use the cost of NFA simulation [\[5\]](#) as the cost of string encoding. Intuitively, the number of states in each state vector denotes the number of choices for each step, which is the encoding cost.

The complexity of the above algorithm is  $O(cn^2)$ , where  $n$  is the length of the string for inference, and  $c$  is the numbers of the strings. Although the algorithm is quadratic w.r.t. the length of input string, it is highly efficient when the length of the string is short.

### 3 Document Clustering

In this section, we show how to cluster XML documents based on their structures using the edit distance between XML documents and schemata.

Based on the edit distance between an ordered tree and regular hedge grammar, we define the structural distance  $\delta_s(d_1, d_2)$  between XML documents  $d_1$  and  $d_2$  as:

$$\delta_s(d_1, d_2) := (\hat{\delta}(d_1, s_2) + \hat{\delta}(d_2, s_1))/2$$

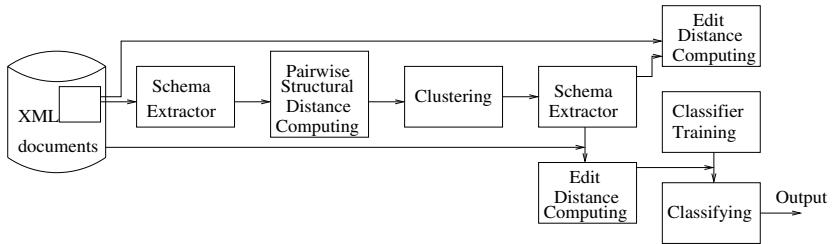
where  $s_1$  and  $s_2$  are the schemata inferred from documents  $d_1$  and  $d_2$  respectively.

To cluster a collection of XML documents, the pairwise structural distances are computed. Each document is represented as a vector:

$$\langle \delta_s(d_i, d_1), \delta_s(d_i, d_2), \dots, \delta_s(d_i, d_n) \rangle,$$

and the distance vectors are fed to a clusterer.

Although pairwise structural distance is very effective in detecting groups of documents, applying it on a large collection of XML documents is difficult. Instead, we use a cluster-classification procedure to handle large number of documents, which can be illustrated by Fig. [\[2\]](#).



**Fig. 2.** Clustering Process for Large Collections of Documents

The advantages of using this distance metric for document clustering are two folds:

1. Once we have the inferred grammar from each class, the representation of each document depends only on the grammars but independent of other documents. So the total number of distances that are needed is linear w.r.t. the number of documents for clustering.
2. The inferred grammar from a collection of documents is likely to contain more structural information than a single document, and the clustering-classification procedure using a grammar to represent a group tends to achieve higher accuracy than using a single document as a representative for a group.

## 4 Implementation and Experimental Results

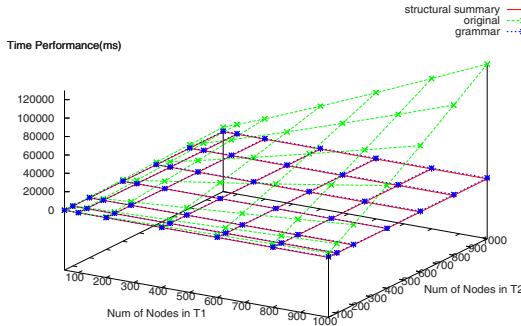
We have fully implemented the algorithms described in the above sections, and developed a prototype system for document clustering.

Based on the implementation, we have tested the clustering system using both real data and synthesized data. The following three datasets are used in our experiments: the Sigmod collection, a synthesized data set from [3], and the MovieDB data set from XML Mining Challenge [9].

To evaluate the time efficiency, we compared our method with the clustering method using edit distance between trees, and the clustering method using edit distance between structural summaries. The time for structural summary method includes time for computing the tree summaries, and the time needed for edit distance between tree summaries. The time for our method includes the time for tree size reduction, schema extraction, and the time needed for computing the edit distance between the tree and the schema. Fig. 3 shows the time performance on a pair of documents of variable sizes from MovieDB dataset.

From Fig. 3, we know that the original tree distance method is the slowest one, and our method is a little slower than the structural summary method, but the difference is small.

The clustering quality is evaluated by following the same procedure as described in [3]. In our experiments, the CLUTO [12] system is used to cluster the



**Fig. 3.** Time performance for original, structural summaries, grammar

vector representations of the documents. The number of true positive, false positive, false negative for each group, and overall *Precision P* are used to compare different methods.

For a collection of documents that are clustered into  $n$  groups,  $C_1, \dots, C_n$  with corresponding DTDs  $D_1, \dots, D_n$ , let  $a_i$  be true positive for cluster  $C_i$ ,  $b_i$  be false positive for cluster  $C_i$ , and  $c_i$  be false negative for cluster  $C_i$ , we have precision:

$$P := \frac{\sum_i a_i}{\sum_i a_i + \sum_i b_i}.$$

The clustering results for different datasets are presented in Tables 1, 2, and 3.

Notice that in our algorithm, the values of  $P$  reach excellent level (better than 95%) for all three sets of data. The structural summary method can produce

**Table 1.** Clustering Results: Sigmod Data

| Cluster No | w/o summaries |   |   | w summaries |   |   | tree grammar |   |   |
|------------|---------------|---|---|-------------|---|---|--------------|---|---|
|            | a             | b | c | a           | b | c | a            | b | c |
| 1          | 19            | 1 | 1 | 20          | 0 | 0 | 20           | 0 | 0 |
| 2          | 19            | 1 | 1 | 20          | 0 | 0 | 20           | 0 | 0 |
| 3          | 20            | 0 | 0 | 20          | 0 | 0 | 20           | 0 | 0 |
| P          | $P = 96.7\%$  |   |   | $P = 100\%$ |   |   | $P = 100\%$  |   |   |

**Table 2.** Clustering Results: Bookstore Data

| Cluster No | w/o summaries |    |    | w summaries  |   |   | tree grammar |   |   |
|------------|---------------|----|----|--------------|---|---|--------------|---|---|
|            | a             | b  | c  | a            | b | c | a            | b | c |
| 1          | 12            | 15 | 8  | 20           | 8 | 0 | 20           | 2 | 0 |
| 2          | 5             | 8  | 15 | 12           | 0 | 8 | 18           | 0 | 2 |
| 3          | 20            | 0  | 0  | 20           | 0 | 0 | 20           | 0 | 0 |
| P          | $P = 61.7\%$  |    |    | $P = 86.7\%$ |   |   | $P = 96.7\%$ |   |   |

**Table 3.** Clustering Results: MovieDB Data

| Cluster No | w/o summaries |    |    | w summaries  |   |   | tree grammar |   |   |
|------------|---------------|----|----|--------------|---|---|--------------|---|---|
|            | a             | b  | c  | a            | b | c | a            | b | c |
| 1          | 8             | 12 | 12 | 14           | 4 | 6 | 20           | 3 | 0 |
| 2          | 7             | 14 | 13 | 15           | 7 | 5 | 20           | 0 | 3 |
| 3          | 10            | 9  | 10 | 14           | 6 | 6 | 20           | 0 | 0 |
| P          | $P = 41.6\%$  |    |    | $P = 71.6\%$ |   |   | $P = 95\%$   |   |   |

very good results when the length of the repeat pattern is 1, but the accuracy becomes significantly degraded (to 71.6%) when the repeat patterns are more complicated.

**Remarks:** The evaluation results indicate the following:

- Regular hedge grammar is a better way to characterize the structural properties of XML documents than structural summary.
- The schema extraction method using MLD principle and NFA simulation cost is effective in extracting schema rules from a collection of documents.
- The time performance of computing edit distance between a tree and a NRHG is suitable for clustering Web documents in real world applications.

## 5 Conclusions

In this paper, we presented a framework for clustering XML documents using *structural distance*. It is based on the definition of the edit distance between an XML document and a schema. We have also covered the validity of using the edit distance, and a novel approach for schema extraction. Although it is more complicated than the methods presented in [8] and [3], it can cluster documents having more complicated structure with much higher accuracy. Experimental studies have shown the efficiency and efficacy of our approaches using both real and synthesized data.

## References

1. N. Suzuki, *Finding an Optimum Edit Script between an XML Document and a DTD*, ACM SAC'05, pp. 647 - 653, March, 2005, Santa Fe, NM.
2. G. Xing, *Fast Approximate Matching Between XML Documents and Schemata*, APWeb 2006 (X. Zhou et al. Eds), LNCS 3841, pp. 425-436, Springer-Verlag, 2006.
3. T. Dalamagas, T. Cheng, K. Winkel, T. Sellis *A methodology for clustering XML documents by structure*, Information Systems, 31(3): 187-228 (2006).
4. K. Thompson, *Regular Expression Search Algorithm*, Communications of ACM, vol 11-6, pp 419-422, 1968.
5. A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

6. D. Shasha, K. Zhang, *Approximate Tree Pattern Matching*, Chapter 14 Pattern Matching Algorithms (eds. Apostolico, A. and Galil, Z.), Oxford University Press, June 1997.
7. M. Murata *Hedge Automata: A Formal Model for XML Schemata* [http://www.xml.gr.jp/relax/hedge\\_nice.html](http://www.xml.gr.jp/relax/hedge_nice.html)
8. A. Nierman, H. V. Jagadish, *Evaluating structural similarity in XML documents*, WebDB 2002, Madison, Wisconsin, June 2002.
9. XML Document Mining Challenge, <http://xmlmining.lip6.fr/>
10. Boris Chidlovskii, *Schema Extraction from XML Data: A Grammatical Inference Approach*, KRDB'01 Workshop, Rome, Italy, September 15, 2001.
11. M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shim, *Xtract: A System for Extracting Document Type Descriptors from XML Documents*, SIGMOD Conference 2000, pp. 165-176, May 16-18, 2000, Dallas, Texas, USA.
12. G. Karypis, *CLUTO A clustering toolkit* Technical Report 02017, University of Minnesota, Department of Computer Science, Minneapolis, MN 55455, Aug. 2002.

# The Multi-view Information Bottleneck Clustering

Yan Gao<sup>1</sup>, Shiwen Gu<sup>1</sup>, Jianhua Li<sup>1</sup>, and Zhining Liao<sup>2</sup>

<sup>1</sup> College of Information Science and Engineering, Central South University,  
410075, Hunan, China

{gaoyan, swgu, jhli}@csu.edu.cn

<sup>2</sup> Department of Computer Science, Loughborough University  
Leics, UK, LE113TU  
Z.Liao@lboro.ac.uk

**Abstract.** In this paper, we propose a new algorithm for information bottleneck method in multi-view setting where instances have multiple independent representations. By introducing the two important conditions, conditional independence and compatibility, into the information bottleneck clustering, the compatible constraint maximizing the agreement between clustering hypotheses on different views is imposed on the individual views to cluster instances. Our algorithm is developed by the compatible constraint. Experiments on three real-world datasets indicate that our algorithm considering the relationship among multiple views can provide solution with improved quality in multi-view setting.

## 1 Introduction

The information bottleneck method [1] (IB) is to extract structure from the data by viewing structure extraction as data compression while conserving relevant information. IB is successfully applied to clustering instances in different domains, such as document clustering [1][2], image segmentation [3] etc. And it is also extended to cluster heterogeneous data [4][5] and to find classes that are in some sense orthogonal to existing knowledge [6][7]. Different from others' work, we want to cluster instances with the information bottleneck in multi-view setting in which instances have multiple independent representations. In multi-view setting, if the information bottleneck method is used to cluster instances in the individual view, the information in other views is omitted. The ensemble [8][9] of the partitions on different views is a good method to cluster the multi-representative instances, but it omits the relationship between the different views.

The co-training method proposed in [10] is a classic semi-supervised algorithm in multi-view setting. The idea of the co-training method is to train one learner on each view of the labeled examples and then to iteratively let each learner label the unlabeled examples predicted with the highest confidence. The important assumptions of co-training are: conditional independence and compatibility.

In this paper, a new algorithm is proposed for the information bottleneck method in multi-view setting. By introducing the two important conditions of co-training: conditional independence and compatibility, into the information bottleneck

clustering, the compatible constraint maximizing the agreement between clustering hypotheses on different views is imposed on the individual views to cluster instances.

## 2 Related Work

### 2.1 Information Bottleneck

With the data modeled by a random variable  $X$ , relevant information is explicitly modeled by a second random variable  $Y$ . The information bottleneck method [1] is to construct a probabilistic clustering, given by a random variable  $C$ , such that the mutual information  $I(X;C)$  between the data and the clusters is minimized, i. e.  $C$  compresses the data as much as possible, while at the same time the mutual information  $I(Y;C)$  of the relevant variable  $Y$  and the clusters is maximized, i. e. the relevant structure is conserved. Both goals are balanced against each other by a real parameter  $\beta > 0$ , the objective function of IB is defined as:

$$\begin{aligned} F &= I(X;C) - \beta I(Y;C) \\ I(X,C) &= \sum_{x,c} p(x,c) \log \frac{p(x,c)}{p(x)p(c)} \end{aligned} \quad (1)$$

### 2.2 Co-training

The co-training method is a classic multi-view semi-supervised algorithm. The important conditions for multi-view learning algorithm are introduced by Blum and Mitchell in the co-training method: conditional independence and compatibility.

**Definition 1 (conditional independence):** In a domain with two views  $V_1$  and  $V_2$ , the instance  $x$  is represented by a triple  $\langle x_1, x_2, l \rangle$ .  $V_1$  and  $V_2$  satisfy conditional independent just in the case: for all  $x_1 \in V_1, x_2 \in V_2, l \in \text{Label}$ ,

$$\begin{aligned} \Pr[X_1 = x_1 | X_2 = x_2, x = l] &= \Pr[X_1 = x_1 | x = l] \\ \Pr[X_2 = x_2 | X_1 = x_1, x = l] &= \Pr[X_2 = x_2 | x = l] \end{aligned} \quad (2)$$

**Definition 2 (compatibility):** In a domain with two views  $V_1$  and  $V_2$ , supposing  $f_1$  is the target concept in  $V_1$ ,  $f_2$  is the target concept in  $V_2$ ,  $f$  is the global target concept,  $V_1$  and  $V_2$  satisfy compatibility just in the case: For most instances  $x = \langle x_1, x_2 \rangle$ ,

$$f(x_1, x_2) = f_1(x_1) = f_2(x_2) \quad (3)$$

“Conditional independence” indicates that it becomes unlikely for compatible classifiers trained on independent views to agree on an incorrect label. “Compatibility” indicates that the label achieved in a single view should agree with the label obtained in other views for most instances.

### 3 The Multi-view Information Bottleneck

#### 3.1 The Multi-view Objective Function and Solution

For multi-view clustering, “conditional independence” and “compatibility” are also important conditions. The first means that we must deduce the correct hypothesis from the clustering hypotheses obtained in the different views. The second means that these clustering hypotheses should agree with each other at a large extent. The mutual information can be used to measure the agreement between the hypotheses in different views. In this section, we analyzed the mutual information between the hypotheses in different views, and proved that maximizing these mutual information can reveal lots of information about correct hypothesis. Based on this deduction, a new objective function for the information bottleneck method in multi-view setting is proposed.

**Proposition 1.** In a domain with multiple views, let  $|V|$  be the number of views,  $C$  be the correct clustering hypothesis,  $C_i$  be the clustering hypothesis in the  $i$ -th view, so be  $C_j$ . There exists a relationship between  $C$  and  $C_i$ ,  $C_j$  as:

$$I(C_i; C) \geq \frac{1}{|V|-1} \sum_{j \neq i} I(C_i; C_j), \quad \forall i, j, 1 \leq i, j \leq |V| \quad (4)$$

**Proof.** In a domain with multiple views, every view is conditional independent, and only depends on the distribution of correct clustering hypothesis  $C$ , so a Markov chain can consist of  $C$  and  $C_i$ ,  $C_j$ :  $C_i > C -> C_j$ . And the inequality (5) can be deduced from the data-processing inequality of information theory [11]:

$$\begin{aligned} I(C_i; C) &\geq I(C_i; C_j) \quad (i \neq j) \\ \Rightarrow (|V|-1)I(C_i; C) &\geq I(C_i; C_1) + \dots + I(C_i; C_{i-1}) + I(C_i; C_{i+1}) + \dots + I(C_i; C_{|V|}) \quad (5) \\ \Rightarrow I(C_i; C) &\geq \frac{1}{|V|-1} \sum_{j \neq i} I(C_i; C_j) \end{aligned}$$

According to Proposition 1, if  $C_i$  and  $C_j$  agree with each other to a large extent, then they must reveal a lot about  $C$ . For clustering instances with multiple representations in the  $i$ -th view, we should maximize the agreement between the hypothesis in the  $i$ -th view and the hypotheses in other views. This is the compatible constraint for every single view. Based on the compatible constraint, the objective function for clustering instances in the  $i$ -th view is described as:

$$\max_{p(c_i|x)} \beta I(Y; C_i) - I(X; C_i) + \frac{\eta}{|V|-1} \sum_{j \neq i} I(C_i; C_j) \quad (6)$$

Because of the conditional independence of every view, the objective function in whole is:

$$\arg \max_{\substack{\{p(c_i|x), \dots, p(c_{|V|}|x)\} \\ \sum_{c_i} p(c_i|x) = 1}} \sum_{i=1..|V|} [\beta_i I(Y_i; C_i) - I(X; C_i)] + \frac{\eta}{|V|-1} \sum_{i=1}^{|V|} \sum_{j>i} I(C_i; C_j) \quad (7)$$

The Lagrangian function for (7) is:

$$\ell = \sum_{i=1..|V|} [\beta_i I(Y_i; C_i) - I(X; C_i)] + \frac{\eta}{|V|-1} \sum_{i=1}^{|V|} \sum_{j>i} I(C_i; C_j) + \sum_{i=1..|V|} \lambda_i(x) \sum_{c_i} p(c_i|x) \quad (8)$$

Computing the partial derivative of the function (8) with respect to the variable  $p(c_i|x)$ , the iterating equation (9)~(12) can be available. Computing these iterative equations, the local maximum solution of (7) is obtained:

$$p(c_i|x) = \frac{p(c_i)}{Z(i)} \exp(-\beta_i D_{ki}[p(y_i|x) \| p(y_i|c_i)]) - \frac{\eta}{|V|-1} \sum_{j \neq i} D_{kj}[p(c_j|x) \| p(c_j|c_i)] \quad (9)$$

$$p(c_i) = \sum_x p(x)p(c_i|x) \quad (10)$$

$$p(y_i|c_i) = \sum_x p(y_i|x)p(x|c_i) \quad (11)$$

$$p(c_j|c_i) = \frac{\sum_x p(x)p(c_j|x)p(c_i|x)}{p(c_i)} \quad (12)$$

Where  $Z(i)$  is normalized factor.

### 3.2 The Global Ensemble

When the set of hypotheses in different views are obtained, the correct hypothesis should be deduced from these hypotheses. For this clustering ensemble problem, A. Strehl[9] proposed an objective function based on mutual information(See Equation 13). Solving objective function (13) with the information bottleneck method too, the final hypothesis can be obtained.

$$C^* = \arg \max_C \sum_{j=1}^{|V|} I(C_j; C) \quad (13)$$

## 4 Experiments

We performed experiments to compare the performance of our algorithm (MVIB) with single-view IB and the traditional clustering ensemble. We use two single-view baselines. The first baseline applies IB to a single feature set. The second baseline applies IB to a concatenation of all views (“concat. views”). The experiments are conducted on three datasets [12]. The first is based on Co-training dataset. The second is based on Webkb where we choose 2124 web pages which have three views: “anchor”, “content” and “url”. The third is artificial dataset which comes from 12 of 20 classes of the well-known newsgroups20 dataset. We randomly select 250

examples for each of 12 newsgroups, which result in a dataset with 1000 examples distributed over four classes having the three-view property. The detail of the artificial dataset is described in Table 1.

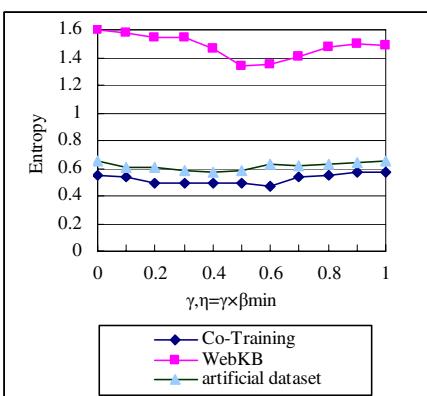
**Table 1.** The detail of artificial dataset based on newgroup20

|         | V1                       | V2                    | V3                 |
|---------|--------------------------|-----------------------|--------------------|
| Class 1 | Comp.os.ms-windows.misc  | talk.politics.misc    | Talk.religion.misc |
| Class 2 | Comp.sys.ibm.pc.hardware | talk.politics.mideast | Talk.politics.guns |
| Class 3 | Alt.atheism              | Rec.sport.hockey      | Soc.religion       |
| Class 4 | Rec.motorcycles          | Rec.sport.baseball    | Misc.forsale       |

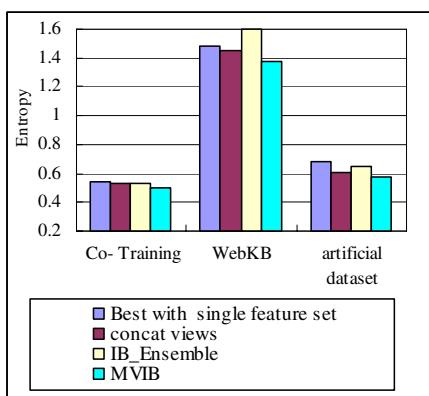
The performance of the three algorithms is measured by the average entropy over all classes (Equation 14).The frequency  $p_{ij}$  counts the number of elements of the  $i$ -th class in  $j$ -th cluster, and  $n_j$  is the size of cluster  $j$ . The low value of entropy indicates the good performance of the clustering algorithm. The results in our experiments are the average entropy over 10 independent runs.

$$E = - \sum_{i=1}^k \frac{n_i}{n} \sum_j p_{ij} \log p_{ij} \quad (14)$$

We use cross-validation test to select the best value of  $\beta$  in every view. Considering preserving more information about terms, the parameter  $\eta$  is not greater than  $\beta_{\min}$ . ( $\beta_{\min} = \min\{\beta_i\}, 1 \leq i \leq |V|$ ). Figure 1 shows the results on three datasets under different value of  $\eta$ . It indicates that MVIB can obtain the lowest value of entropy when  $\eta$  is equal to or near  $\beta_{\min}/2$ . Figure 2 shows the results with three algorithms on three datasets. It indicates the entropy of MVIB is lower than that of single-view IB and that of clustering ensemble. It indicates that considering the compatible constraint can improve the performance of clustering the instances with multiple representations.



**Fig. 1.** The results on three datasets under different value of  $\eta$



**Fig. 2.** The results with three algorithms on three datasets

## 5 Conclusion

We propose a new multi-view information bottleneck algorithm(MVIB) that extends information bottleneck algorithm to multi-view setting to cluster multi-representative instances. By maximizing the clustering hypothesis in different view, we can get a set of clustering hypotheses that reveal lots of information about the correct hypothesis and deduce the final clustering hypothesis from these hypotheses. This allows one to incorporate all available information to form the best clusters when there is lots of single-view data to be clustered. Experiments on three real-world data sets indicate that MVIB that considers the relationship between different views can improve the performance of clustering the instances with multiple representations.

## References

1. Noam Slonim. The Information Bottleneck: Theory and Applications. PhD thesis, Hebrew University, Jerusalem, Israel, 2002.
2. Noam Slonim, NIR friedman, Naftali Tishby. Unsupervised Document Classification using Information Maximization, Proc.25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Tampere, Finland, 2002, pages: 129-136
3. Jacob Goldberger, Hayit Greenspan and Shiri Gordon. Unsupervised Image Clustering using the Information Bottleneck Method, DAGM-Symposium, Zurich, Switzerland 2002. pages:158-165
4. Dhillon, I. S., Mallela, S., & Modha, D. S. Information-theoretic co-clustering. Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, pages: 89-98.
5. Ron Bekkerman, Ran El-Yaniv, Andrew McCallum, Multi-Way Distributional Clustering via Pairwise Interactions, In Proceedings of the 22 International Conference on Machine Learning, Bonn, Germany, 2005, pages: 41-48
6. D. Gondek and T. Hofmann. Non-redundant data clustering. In Proceedings of the Fourth IEEE International Conference on Data Mining, Brighton, UK, 2004, pages: 75–82
7. David Gondek, Thomas Hofmann, Non-Redundant Clustering with Conditional Ensembles, Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, , Chicago, Illinois, USA, 2005, pages:70-77
8. A.Topchy, A.K.Jain, and W. Punch. Combining multiple weak clusterings. In Proceedings of the Third IEEE International Conference on Data Mining, Florida, USA 2003, pages:331-338.
9. Strehl, A. and Ghosh, J. Cluster Ensembles – A Knowledge Reuse Framework for Combining Multiple Partitions. Journal on Machine Learning Research, volume 3, (2002), pages:583-617
10. Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In Annual Conference on Computational Learning Theory (COLT-98), Madison, USA,1998. pages:92-100
11. Thomas M. Cover, Joy A. Thomas, The elements of information theory, China Machine Press, 2005, page: 24.
12. Co-training, Webkb, newsgroup20. <http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20>

# Web Service Composition Based on Message Schema Analysis\*

Aiqiang Gao<sup>1</sup>, Dongqing Yang<sup>1</sup>, and Shiwei Tang<sup>2</sup>

<sup>1</sup> School of Electronics Engineering and Computer Science, Peking University,  
Beijing, 100871, China  
`{aqgao,ydq}@db.pku.edu.cn`  
<sup>2</sup> `tsw@pku.edu.cn`

**Abstract.** In current work for web service composition, the schema and structure of XML messages are abstracted and the messages are assumed to be flat, which is not the case as far as current standards and specifications are concerned. This paper proposes a method for web service composition based on message schema analysis. This method starts from message schema matching between two web services. The concepts of composable web services and composition context are defined to guide the synthesizing process. To perform message schema analysis, a method for MSL schema matching is discussed, where MSL is a formal model for XML Schema. Some usually used web service composition examples are collected from the literature to evaluate the method in this paper.

## 1 Introduction

In current works for automatic composition and verification([1][2]), it is usually assumed that there are a fixed set of web services and the synthesis task is to generate a composite service with respect to this fixed set of services. However, it is possible that the services that may be useful for a composition goal will not be determined a priori. Thus, web service discovery is required.

For web service discovery method such as [3] and [4], the structure of XML messages is assumed to be flat, which is not the case as far as current standards and specifications are concerned(such as WSDL[5] and BPEL[6]).

This paper proposes a method for web service discovery and composition based on message schema analysis. This method starts from message schema matching between two web services. According to the result of message schema matching, the concepts of composable web services and web service composition context are defined to guide the process of web service synthesizing. With the initial input message being specified, this method accomplishes web service composition semi-automatically.

Some usually used web service composition examples are collected from current research works and are used to evaluate the method in this paper.

---

\* This work is supported by the National Natural Science Foundation of China under grant No. 90412010 and 60642004, and the IBM University Joint Research Proposal.

The method in this paper can solve the problem of web service synthesizing at XML messaging level. The main contributions of this paper are:

1. Proposes an approach for synthesizing web service with service discovery mechanism supported. It takes the schema and structure of XML message into account, which is usually neglected by current works.
2. Discusses a method for matching two MSL schema. The matching results serve as the base for interpreting and executing composite web services.

The rest of this paper is organized as follows. Section 2 describes MSL(Model Schema Language) and MSL-based schema matching method; Section 3 presents a method for synthesizing web services; Section 4 discusses implementation details and gives some composition scenarios; Section 5 reviews related works and Section 6 concludes this paper and discusses future work.

## 2 MSL-Based Schema Matching Method

Model Schema Language(MSL) is an attempt to formalize some of the core ideas in XML Schema. The benefits of a formal description is that it is both concise and precise. MSL uses a mathematical notation that is easier to manipulate formally than the XML syntax of Schema. Mathematical notation is used for model groups, components, and documents. In this paper, a concise version of MSL is introduced, which includes sequence, choice and multiple-occurrence.

**Definition 1. (MSL)**

$g \rightarrow b|t[g_0]|g_1\{m,n\}|g_1, \dots, g_k|(g_1|\dots|g_k)$  where  $b$  is a basic data type,  $t$  is a tag name,  $m$  and  $n$  are two integer variables. The meaning is:

- (1)  $t[g_0]$  denotes an element, the tag of whose root element is  $t$ , the type of its child is compatible with  $g_0$
- (2)  $g_1\{m,n\}$  is a sequence of elements with type  $g_1$ , with min occurrence and max occurrence being  $m$  and  $n$ , respectively
- (3)  $g_1, \dots, g_k$  is a list of elements whose type is  $g_1, g_2, \dots, g_k$ , respectively
- (4)  $g_1|\dots|g_k$  is a choice between type  $g_1, g_2, \dots, g_k$ , where  $g_i$  is gotten from  $g \rightarrow b$  or  $g \rightarrow t[g_0]$

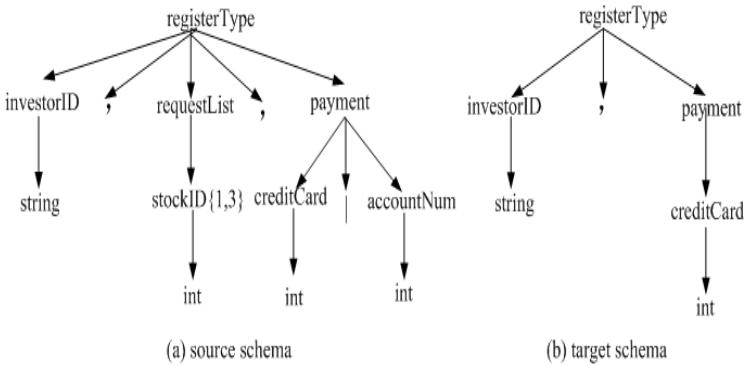
Given MSL denotation, parsing process read this denotation and identify key symbols like “{”, “}”, “[”, “]”, “,” and “|”, then grammar-directed translation procedure is adopted to translate the syntax into a tree structure. The schema tree will be used by matching method to get the compatibility between two schemas.

There are usually shared components in XML schema. Because XML Schema in web service is relatively simpler, materialization-based method for shared components is adopted in this paper.

For schema matching, the matching method starts from matching between leaf nodes because leaf nodes contain the main content. The mapping for lower level nodes are spread to upper levels. Once a couple of nodes is found to be matching, the paths from root to the nodes in source schema tree and target schema tree are checked. The checking has two effects: to ensure these two nodes

are indeed matching with respect to their path context and to record the path as the query expression for data retrieval.

The matching method based on MSL schema tree will be used by the synthesizing method to check the compatibility of two web services.



**Fig. 1.** An example for schema matching

For schemas shown in Fig. 1, the element *investorID* in target schema will be found to be matching with element *investorID* in source schema. And, element *creditCard* in target schema will be found to be consistent with element *creditCard* in source schema.

### 3 SMSM Method for Web Service Synthesizing

Before presenting SMSM method(web service Synthesizing method based on Message Schema Matching), we will discuss several concepts first.

The abstract process of a composite web service is represented using  $G = (V, E; F, Q)$ , where

(1)  $V = \{T_1, T_2, \dots, T_n\}$ , each task  $T_i$  is defined by a web service type and its format is *servicename+operationname*, such as  $\langle \text{task}, \text{operation} \rangle$ .

(2) each edge  $e \in E$  represents the transition between task  $T_i$  and  $T_j$

(3)  $F : E \rightarrow REL$  is a mapping that associates each edge  $e \in E$  a control transition relationship, where  $REL = \{\text{SEQ}, \text{AND-S}, \text{AND-J}, \text{OR-S}, \text{OR-J}\}$ , representing sequence, parallel split, parallel join, conditional split and conditional join, respectively. Now, only three control structures are supported: Sequence, Parallel and Conditional.

(4)  $Q : E \rightarrow \text{Query}$  is a mapping that associates each edge  $e \in E$  a rule for transforming data, which is the result of schema matching discussed in last section.

A web service composition context  $C$  is a pair like  $\langle MS, G \rangle$ , where  $MS = \{MS_1, MS_2, \dots, MS_k\}$ , each  $MS_i$  representing a XML message schema, and  $G$  is the current abstract process of a composite web service.

In the following paragraphs,  $\text{Composable}(A, B) = \text{True}$  is used to denote that  $A$  and  $B$  can be combined together. The composable of two web services is checked according to the schema matching method in last section.

### Definition 2. Executable web service

The fact that a web service  $ws$  is executable in a web service composition context  $C = (MS, G)$  is denoted as  $\text{executable}(ws, C)$ , which is defined as:  $(\exists ws' \in V(\text{Composable}(ws', ws) = \text{True})) \vee (ws.\text{input} = \pi_{ws.\text{input}}(MS))$ , where the first part denotes that there exists web service type  $ws' \in V$  satisfies  $\text{Composable}(ws', ws) = \text{True}$ ; the second part denotes that although there is not such a  $ws'$ , the input message of  $ws$  can be satisfied by current message set  $MS$ , so this service is also defined as executable.

According to the above discussion, web service synthesizing method **SMSM** is shown in Algorithm 1. When the client specifies input and the expected output, the algorithm will check the satisfaction of the output recursively. At each step, if the output is not satisfied, the executable services under current context are discovered and one is selected to add into  $G$ . This process repeats until we get the expected output. During this process, the client needs to participate in specifying input and making decision when several services are identified as candidates in current composition context.

---

#### Algorithm 1. Semi-Automatic Synthesizing Algorithm

---

```

1: Input: $inMsg, outMsg$;
2: Output: $G = (V, E; F, Q)$;
3: $G = (V, E; F, Q)$ with $V = \emptyset, E = \emptyset$;
4: $MS = \{inMsg\}, C = (MS, G)$;
5: while ($outMsg$ is not satisfied from MS) do
6: $currServiceSet = serviceDiscovery(C)$; \triangleright each s in $currServiceSet$ is
 executable under C , i.e., $\text{executable}(s, C) = \text{True}$
7: $currService$ is the service selected from $currServiceSet$;
8: $MS = MS \cup \{currService.\text{output}\}$;
9: $V = V \cup \{currService\}$;
10: $E = E \cup \{e\}$, where $e = (s', currService)$, with each s' linked to $currService$;
11: $F(e)$ is assigned the relationship associated with edge e ;
12: $Q(e)$ is assigned the mapping expression for $currService.\text{input}$;
13: end while
14: store G into process database;
15: new a web service for this process G and publish it;

```

---

## 4 Examples

The method discussed here is implemented in SOSE framework for Synthesizing, Optimal Selection and Execution of composite web services. The framework includes two parts that correspond to two phases in the life cycle of a composite web service. During the synthesizing phase, a composite web service will be generated and denoted as an abstract business process. Then, the process is

deployed and executed in the second phase where an optimal execution plan is generated and performance analysis is conducted. The details of this framework are not discussed here due to space limitation.

In our work, some examples have been discussed for Algorithm 1. The results illustrate that SMSM can generate composite web services according to message schema effectively. Those examples include loan approval and purchase order in [6], flower shop in [8], travel planning in [9] and car broker in [10].

Flower delivery([8]) example is discussed briefly to illustrate the SMSM method. Initially, the input XML message includes the following elements: *personname, personname, flowername, numofflower* and *creditcard*. The SMSM method searches the web services set and selects *Directory*(mapping name to address) service as the initial service because it is executable according to the initial context. Then it will select the flower shop service(flower ordering) because the flower name, the number of flower and the target are all known. After the flower is ordered, the *CreditCard* service will be added into the composition service to pay for the ordering. With the authorization message gotten from the *CreditCard* service, the *Dispatch* service becomes executable in current composition context and is combined into the composite service. In the end, the reply message is formed and sent to users.

## 5 Related Works

Web service composition ([1]) is to build value-added services and web applications by integrating and composing existing elementary web services. In current works for automatic composition and verification([2][11]), the composite web service is generated according to a fixed set of web services. However, it is probable that the composite web service should be synthesized from a large set of web service with some irrelevant ones. Thus, it is necessary to consider web service discovery mechanism.

For web service discovery method such as [3] and [4], the schema and structure of XML messages are abstracted and assumed to be flat, which is not the case as far as current standards and specifications are concerned.

The method in this paper for web service composition takes the schema and structure of XML message into account. The method accomplishes the composition task with respect to web service discovery mechanism. It avoids the limitation that target service is synthesized with a fixed set of services. The service discovery method can handle the schema and structure of XML message, which considers XML-based web service protocol stack in practice.

Schema matching methods have been discussed for years,such as [12][13][14]. Authors in [12] give a good survey where the methods are categorized according to several criteria. Our method is similar to [13] which is schema oriented.

## 6 Conclusion

In this paper, a method for web service composition at message level is proposed. To accomplish this task, a method for MSL schema matching is first discussed,

where MSL schema is a formal data model for XML Schema. According to the result of schema matching, two concepts of composable web service and composition context are defined to guide the synthesizing process. In future works, more XML schema components and examples are needed to evaluate this method. And the verification of composite web services is also one of the further works.

## References

1. R. Hull, M. Benedikt, V. Christophides, and J. Su.E-services: A look behind the curtain. In Proc. ACM Symp.on Principles of Database Systems, 2003.
2. Giuseppe De Giacomo,Daniela Berardi and Massimo Mecella,Basis for Automatic Web Service Composition, tutorial at WWW2005, available at <http://www.dis.uniroma1.it/~degiacom/>
3. B. Benatallah, M. S. Hacid, A. Leger, C. Rey and F. Toumani. On automating Web services discovery. the VLDB Journal 14(1): 84-96,2005
4. Z. Shen and J. Su. Web Service Discovery Based on Behavior Signatures. In Proceedings of IEEE International Conference on Services Computing (SCC) 2005.
5. W3C, "Web Services Description Language (WSDL) Version 2.0", W3C Working Draft, March 2003. (See <http://www.w3.org/TR/wsdl20/>.)
6. Business Process Execution Language for Web Services, version 1.1, <http://www.ibm.com/developerworks/library/ws-bpel/>
7. A. Brown, M. Fuchs, J. Robie and P. Wadler. MSL a model for W3C XML Schema. In Proceedings of the 10th International Conference on World Wide Web(WWW),2001, 191-200.
8. A. Kumar, S. Mittal and B. Srivastava. Information Modeling for End to End Composition of Semantic Web Services. In IBM Research Report RI05001, 2005,<http://domino.watson.ibm.com/library/CyberDig.nsf/Home>
9. Liangzhao Z., Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant K., Henry Ch., QoS-Aware Middleware for Web Services Composition, IEEE transactions on Software Engineering , 2004,30(5):311-327
10. B. Medjahed, A. Bouguettaya and A. K. Elmagarmid. Composing Web services on the Semantic Web. VLDB Journal 12(4),2003
11. D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini and M. Mecella. Automatic Composition of E-services That Export Their Behavior. In Proceedings of International Conference on Service Oriented Computing(ICSOC),2003, 43-58.
12. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. The VLDB Journal 10(4),2001.
13. J. Madhavan, P. A. Bernstein and E. Rahm. Generic Schema Matching with Cupid. In Proceedings of the 27th International Conference on Very Large Databases(VLDB),2001, Roman,Italy.
14. D. Aumueller, H.-H. Do, S. Massmann and E. Rahm. Schema and Ontology Matching with COMA++. In Proceedings of the ACM SIGMOD International Conference on Management of Data,2005, 906-908.

# SQORE: A Framework for Semantic Query Based Ontology Retrieval

Chutiporn Anutariya<sup>1</sup>, Rachanee Ungrangsi<sup>1</sup>, and Vilas Wuwongse<sup>2</sup>

<sup>1</sup> School of Technology, Shinawatra University

99 Moo 10 Bangtoey, Samkok, Pathum Thani, 12160 Thailand

{chutiporn, rachanee}@shinawatra.ac.th

<sup>2</sup> School of Engineering and Technology, Asian Institute of Technology

P.O. Box 4, Klong Luang, Pathum Thani, 12120 Thailand

vw@cs.ait.ac.th

**Abstract.** Existing approaches to ontology retrieval solely base their search mechanisms on keyword matching while taxonomic structure is solely used for ranking purpose. Users, therefore, are not equipped with expressive means to structurally and semantically describe their ontology needs. To tackle this problem, this paper develops a framework for *Semantic Query based Ontology Retrieval*, namely *SQORE*. It enables precise formulation of a *semantic query* in order to best capture a user's ontology requirements, which include not only the desired class and property names, but also their relations and restrictions. *SQORE* employs *XML Declarative Description (XDD) theory* as its theoretical foundation for modeling ontology databases and evaluating semantic queries. Moreover, *similarity score* is formally defined as a key metric for ranking the resulting ontologies based on their conceptual closeness to the given query.

## 1 Introduction

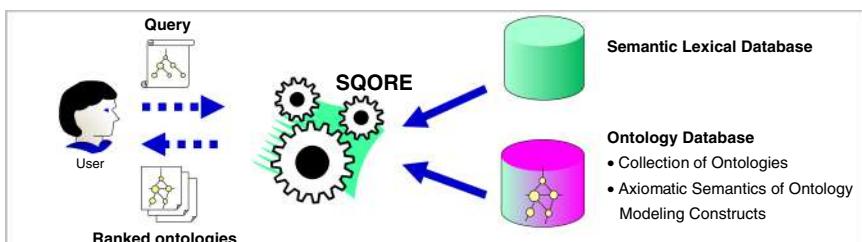
The *Semantic Web* aims to expand the World Wide Web by allowing data to be shared and reused across applications and communities via *ontology* [4]. However, existing ontology search engines primarily base their approaches on search terms which cannot sufficiently capture the structural and semantic information about the domain concepts that users want. *Swoogle* [6, 7] and *OntoKhoj* [9] implement the *Google's PageRank*-like algorithm [5] which creates the rankings based on ontology referral network. However, this approach is currently inefficient due to the lack of links among ontologies on the Web. *OntoSearch* [10] provides several criteria for users to evaluate and browse the ontologies and then uses *AKTiveRank* [1] as metrics for ontology ranking based on the taxonomic structure information such as class names, shortest paths, the linking density and the positions of focused classes in the ontology. Although a large number of ontologies are returned as the result in these approaches, they are often un-useable because they do not meet user requirements or otherwise they require tremendous modification efforts. Furthermore, semantic information such as properties and ontological relations (e.g. `subClassOf`, `inverseOf`, `equivalentClass`, and `subPropertyOf`) is not considered.

This paper develops *SQORE – Semantic Query based Ontology Retrieval* framework which allows users to structurally and semantically formulate their ontology requirements in terms of *semantic queries*. It employs *XML Declarative Description (XDD) theory* [2, 3, 11] as its theoretical foundation for modeling ontology databases and evaluating semantic queries, which does not only facilitate ontology matching and retrieval, but also support reasoning capability to enhance the matching results. Moreover, it also enables the use of a semantic lexical database, such as *WordNet* [8], for determining semantic relation between two given terms. Thus, the retrieval performance (precision and recall) can be significantly improved when compared to a conventional keyword search. In addition, it employs *similarity score* to rank the resulting ontologies by focusing on their conceptual closeness to the formulated semantic query.

Sect. 2 presents SQORE's overview architecture. Sect. 3 describes ontology database modeling and its semantics. Sect. 4 develops an approach to semantic query formulation and evaluation. Sect. 5 concludes and draws future research directions.

## 2 SQORE's System Architecture Overview

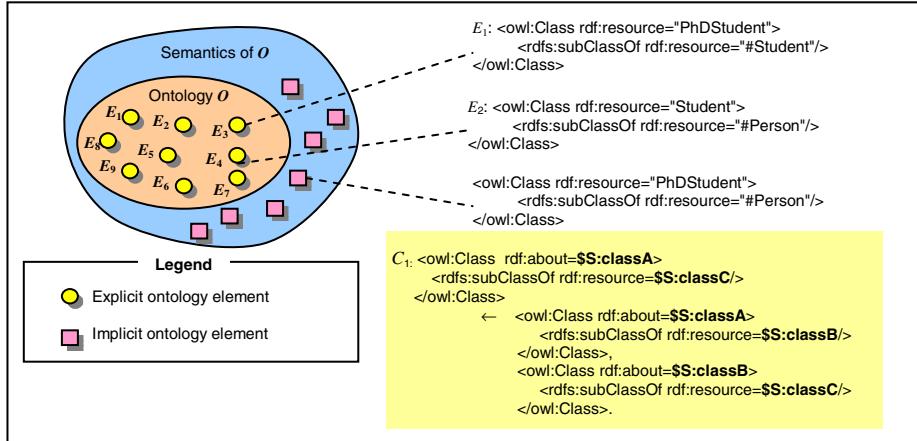
Fig. 1 illustrates SQORE's system architecture which comprises four major components: i) a *semantic query*, ii) a *retrieval engine*, iii) an *ontology database*, and iv) a *semantic lexical database*. In essence, the system works as follows: First, a user formulates and submits a semantic query which precisely captures his/her ontology requirements. The system then executes such query by semantically evaluating it against the ontology database, which comprises a collection of ontologies and a set of rules defining ontology axiomatic semantics. By incorporating these rules, implicit information about classes/properties in a query and an ontology can be derived, and hence enabling semantic query evaluation. Furthermore, when class/property names defined in a query and an ontology do not *exactly match* ( $=$ ), four possibilities occur: i) *equivalence* ( $\equiv$ ): the two terms are synonym, ii) *more general* ( $\supseteq$ ): the query term is broader, iii) *less general* ( $\subseteq$ ): the ontology term is broader, and iv) *unknown* ( $\neq$ ): the relation is unknown. To tackle this, a referenced lexical database, such as *WordNet* [8], is employed in order to determine their appropriate semantic relation. Finally, the system computes the *semantic similarity score* between a given query and an ontology in the collection, which ranges from 0 (strong dissimilarity) to 1 (strong similarity), and returns as the answer the list of ranked ontologies.



**Fig. 1.** SQORE System Architecture

### 3 Ontology Database Model and Its Semantics

By employment of XDD theory, an ontology formalized in RDF(S) or OWL can readily and directly become an XDD description. In order to determine the meaning of a particular ontology, formalization of the axiomatic semantics of each RDF(S)/OWL modeling construct is demanded.



**Fig. 2.** Semantics of the ontology  $O$  wrt. the axiomatic semantics of RDF(S)/OWL constructs

Fig. 2 gives an example of formalizing `rdfs:subClassOf` in terms of an XDD description  $C_1$ . By incorporating this description, the meaning of the ontology  $O$ , consequently, yields elements  $E_1-E_9$  which are explicitly defined by  $O$ , together with the derived ones. For instance, as depicted by Fig. 2, assume that Query  $Q$  searches for ontologies with `PhDStudent` modeled as a `subClassOf` `Person`, while Ontology  $O$  defines `PhDStudent` as a `subClassOf` `Student` which is then a `subClassOf` `Person`; therefore by modeling the semantics of `subClassOf` to be transitive, one can derive that in Ontology  $O$ , `PhDStudent` is also a `subClassOf` `Person`, and thus satisfying the query requirement.

Founded on this formalism, an ontology database  $ODB$  becomes an XDD description with two parts: i)  $ODB_C$ : An ontology collection consisting of  $n$  ontologies  $O_1, \dots, O_n$ , and ii)  $ODB_A$ : A set of XML clauses defining the axiomatic semantics of ontology modeling constructs. The database semantics, denoted by  $\mathcal{M}(ODB)$ , are the set of OWL elements describing classes, properties or instances (individuals) which are directly represented by the database via the ontology collection  $ODB_C$  plus those derivable from it via the axiomatic semantics  $ODB_A$ .

### 4 Semantic Query Formulation and Evaluation

**Definition 1 (Semantic Query).** A *semantic query* is formulated as an XML element of the form:

```

<sq:Query>
 <sq:mandatoryConditions> $m_1 \ m_2 \dots \ m_n$ </sq:mandatoryConditions>
 <sq:optionalConditions> $o_1 \ o_2 \dots \ o_r$ </sq:optionalConditions>
 <sq:semanticLexiconReference> url </sq:semanticLexiconReference>
 <sq:similarityWeightFactors>
 <sq:mandatoryConditionWeight> w_M </sq:mandatoryConditionWeight>
 <sq:stringMatching> $w_=$ </sq:stringMatching>
 <sq:synonymRelation> $w_=$ </sq:synonymRelation>
 <sq:moreGeneralRelation> w_{\supseteq} </sq:moreGeneralRelation>
 <sq:lessGeneralRelation> w_{\subseteq} </sq:lessGeneralRelation>
 <sq:unknownRelation> w_{\neq} </sq:unknownRelation>
 </sq:similarityWeightFactors>
</sq:Query>

```

where

- $m_i$  is an OWL/RDF(S) expression describing a query's mandatory condition,
- $o_i$  is an OWL/RDF(S) expression describing a query's optional condition,
- $url$  gives a URL of a semantic lexical database,
- $w_M, w_=, w_{\supseteq}, w_{\subseteq}, w_{\neq} \in [0, 1]$  are semantic weight factors,
- semanticLexiconReference- and similarityWeightFactors-elements are optional.

Given a semantic query  $Q$ , let  $mcond(Q)$  and  $ocond(Q)$ , respectively, denote the sets of mandatory conditions and optional conditions.  $\square$

Formally speaking, an ontology  $O$  in  $ODB$  satisfies a condition  $m_i$  or  $o_i$  if such a conditional element is included in the meaning of  $O$ . The weight factor  $w_M$  allows explicit specification of how important the mandatory conditions are, and hence  $1-w_M$  becomes the weight for the optional conditions. The semantic lexicon reference specifies external knowledge used for determining appropriate semantic relations between elements of  $Q$  and  $O$ . Thus, based on the discovered semantic relation between a query element  $t_1$  and an ontology element  $t_2$  defined as follows, the weight factors  $w_=, w_{\supseteq}, w_{\subseteq}, w_{\neq}$ , respectively, allow the user to quantify how similar the two elements are. In principle, it is recommended that  $1 = w_{\supseteq} \geq w_= \geq w_{\subseteq} \geq w_{\neq} = 0$ . In practice, these weights can be configured as default settings of the system or manually defined by a user.

## 5 Similarity Measures

This section formally defines important similarity measures. First, let  $\Sigma$  be an *ontology alphabet* comprising *ontology elements* in the following sets:

- $C$ : Class names (such as Person, Student, PhDStudent),
- $P$ : Property names (such as firstname, lastname, supervises, isSupervisedBy),
- $D$ : Datatypes (such as xsd:string, xsd:nonNegativeInteger),
- $M$ : Modeling constructs of RDF(S) and OWL (such as rdfs:subClassOf),
- $R$ : Restrictions on classes and properties, having the form  $m(a,b)$  where  $m \in M$ ,  $a \in (C \cup P)$ ,  $b \in (C \cup P \cup D)$  (such as rdfs:subClassOf(Student, Person)).

The following definition first measures how well two ontology elements match.

**Definition 2 (Element Similarity Score:  $SS_E$ ).** The similarity of two ontology elements  $x$  and  $y$  is measured by:

- For  $x, y \in C$  or  $x, y \in P$  or  $x, y \in D$ :

$$SS_E(x, y) = \begin{cases} w_=& : \text{if } x = y \\ w_\equiv & : \text{if } x \equiv y \\ w_\supseteq & : \text{if } x \supseteq y \\ w_\subseteq & : \text{if } x \subseteq y \\ w_\neq & : \text{otherwise} \end{cases} \quad (1)$$

- For  $x = m(a_1, b_1) \in R$  and  $y = m(a_2, b_2) \in R$ :

$$SS_E(x, y) = SS_E(a_1, a_2)^* SS_E(b_1, b_2) \quad (2)$$

- Otherwise:

$$SS_E(x, y) = \text{unknown} \quad (3) \quad \square$$

Based on this element similarity score definition, one can measure the similarity between a given element  $x$  in  $\Sigma$  and an ontology  $O$  in  $ODB$  by finding the highest similarity score between  $x$  and each element  $y$  that is semantically defined by  $O$ . This is defined by:

**Definition 3 (Best Element Similarity Score:  $SS_B$ ).** The similarity between an element  $x \in C \cup P \cup R$  and an ontology  $O \in ODB$  is measured by:

$$SS_B(x, O) = \max_{y \in M(O)} S(x, y) \quad (4) \quad \square$$

Next, the satisfaction scores of mandatory conditions ( $SS_M$ ) and of optional conditions ( $SS_O$ ) are defined.

**Definition 4 (Satisfaction Scores of Mandatory conditions:  $SS_M$  and Optional conditions:  $SS_O$ ).** With respect to an ontology  $O$ , the satisfaction scores of  $Q$ 's mandatory conditions ( $SS_M$ ) and  $Q$ 's optional conditions ( $SS_O$ ) are measured by:

$$SS_M(Q, O) = \begin{cases} 1 & : \text{if } mcond(Q) = \emptyset \\ 0 & : \text{if } \exists(c \in mcond(Q)) SS_B(c, O) \leq w_\neq \\ \frac{\sum_{c \in mcond(Q)} SS_B(c, O)}{|mcond(Q)|} & : \text{otherwise} \end{cases} \quad (5) \quad \square$$

$$SS_O(Q, O) = \begin{cases} 1 & : \text{if } ocond(Q) = \emptyset \\ 0 & : \text{if } SS_M(Q, O) = 0 \\ \frac{\sum_{c \in ocond(Q)} SS_B(c, O)}{|ocond(Q)|} & : \text{otherwise} \end{cases} \quad (6) \quad \square$$

Finally, one can measure the similarity between a semantic query  $Q$  and an ontology  $O$  by integrating the two components: mandatory condition satisfaction score and optional condition satisfaction score, as follows:

**Definition 5 (Query-Ontology Similarity Score).** The similarity between a semantic query  $Q$  and an ontology  $O$  is measured by:

$$SS(Q, O) = w_M SS_M(mcond_Q, O) + (1 - w_M) SS_O(ocond_Q, O)$$

(7)



## 6 Conclusions

This paper proposes and develops *SQORE* – a novel framework for ontology retrieval system based on *semantic query*. It enables a user to precisely and structurally formulate their ontology requirements, which include not only the desired class and property names, but also their relations and restrictions. Moreover, when evaluating a query, its semantics together with an ontology's semantics are also taken into account in order to correctly and semantically match them.

## Acknowledgement

This work was supported by Thailand Research Fund and Commission on Higher Education, Thailand, under grant number MRG4780192.

## References

1. Alani, H. and Brewster, C.: Metrics for Ranking Ontologies. In Proceedings of 4th International EON Workshop, 15th Int'l WWW Conference, Edinburgh, 2006.
2. Anutariya, C., Wuwongse, V. and Akama, K.: XML Declarative Description with First-Order Logical Constraints. Computational Intelligence, Vol. 21, No. 2, pp. 130-156 (2005)
3. Anutariya, C., Wuwongse, V., Akama, K., Wattanapailin, V.: Semantic Web Modeling and Programming with XDD. Proc. Semantic Web Working Symposium (SWWS-01), CA (2001), 161–180.
4. Berners-Lee, T., Handler, J., and Lassila, O.: The Semantic Web, Scientific American, May 2001.
5. Brin, S., and Page, L. The anatomy of a large-scale hyper-textual web search engine. In Seventh International WorldWide Web Conference, Brisbane, Australia, 1998
6. Ding, L., Finin, T., Joshi, A., Pan, R., Scott Cost, R., Peng, Y., Reddivari, P., Doshi, V., Sachs, J., Swoogle: a search and metadata engine for the semantic web. Proc. 13 ACM Int'l Conf. Information and Knowledge Management, November 08-13, 2004, DC
7. Finin, T., Mayfield, J., Joshi, A., Scott Cost, R., Fink, C.: Information Retrieval and the Semantic Web, Proc. 38th Annual Hawaii Int'l Conf. System Sciences (HICSS'05) - Track 4, p.113.1, 2005
8. Miller, A., Wordnet: A lexical database for English. In Communications of the ACM, number 38(11), 1995.
9. Patel, C., Supekar, K., Lee, Y., Park, E. K., OntoKhoj: a semantic web portal for ontology searching, ranking and classification, Proc. 5th ACM Int'l Workshop on Web Information and Data Management, November 07-08, 2003, Louisiana.
10. Thomas, E., Alani, H., Sleeman, D. and Brewster, C.: Searching and Ranking Ontologies on the Semantic Web. In Proc. Workshop Ontology Management: Searching, Selection, Ranking, and Segmentation. 3rd K-CAP 2005, pp. 57-60, Banff, Canada
11. Wuwongse, V., Anutariya, C., Akama, K., Nantajeewarawat, E.: XML Declarative Description (XDD): A Language for the Semantic Web. IEEE Intelligent Systems, Vol. 16, No. 3, pp. 54–65 (May/June 2001)

# Graph Structure of the Korea Web<sup>\*</sup>

In Kyu Han, Sang Ho Lee, and Soowon Lee

School of Computing  
Soongsil University, Korea  
`{ikhan, shlee}@comp.ssu.ac.kr`

**Abstract.** The study of the Web graph not only yields valuable insight into Web algorithms for crawling, searching and community discovery, and the sociological phenomena that characterize its evolution, but also helps us understand the evolution process of the Web. In this paper, we report the experiments on properties of the Korea Web graph with over 116 million pages and 2.7 billion links. This paper presents the power law distributions from the Korea Web and then compares them with other web graphs. Our analysis reveals that the Korea Web graph has different properties in comparison with the other graphs in terms of the structure of the Web.

**Keywords:** Web graph, Power law, Web evolution, Web connectivity.

## 1 Introduction

The Web can be represented by a directed graph where nodes stand for Web pages, and edges stand for hyperlinks among pages. This graph is likely to have approximately billion nodes as of today, and it is rapidly growing at the rate of 7.3 million new pages a day [15]. There are mathematical, sociological and commercial reasons for studying the evolution of this graph. Exploitation of the information in the graph is helpful for improvement of algorithms for web search, topic classification, cyber-community enumeration and so on.

In the literature, there have been a number of researches regarding the Web graph and power law distributions found on the Web. Albert et. al. [1] estimated the diameter of the Web. Broder and Kumar [2][8] reported the in-degree and out-degree distributions of the global Web graph, which follows the power law distributions, and they also analyzed the structure of the global Web graph. Boldi et. al. [6] studied the African Web. J. Han et. al. [3] showed that the China Web graph manifests properties different from global Web graph. The giant strongly connected component (SCC) of the China Web graph is proportionally bigger than that of the global Web graph is.

Albert et. al. [1], Broder et. al. [2], and Kumar et. al. [13] studied the degree distribution of nodes in the Web graph. They performed empirical studies using graphs of sizes ranging from 325,729 nodes (University of Notre Dame) [1] to 203 million nodes (AltaVista crawler data) [2]. They found that both the in-degree and out-degree of nodes on the Web follow the power-law distributions. The number of

---

<sup>\*</sup> This work was supported by Korea Research Foundation Grant (KRF-2006-005-J03803).

Web pages having a degree  $i$  is proportional to  $1/i^k$  where  $k > 1$ . This implies that the probability of finding a node with a large degree is small yet significant. G. Liu et. al. [4] showed that the China Web has an in-degree distribution with exponent 2.05 and an out-degree distribution with exponent 2.62.

The Web graph is likely to consist of hundreds of millions of nodes and billions of edges. Due to this gigantic scale of the web graph, we can hardly load the full graph into the main memory for enumerating SCCs in Web graph. To solve this problem, J. Han et. al. [3] proposed an algorithm for enumerating SCCs in the Web graph under the split-merge approach. The basic idea of this algorithm is to split the original graph into parts that are smaller enough to load into the main memory, to decompose them one by one, and finally to merge them together.

In this paper, we investigate a number of power law distributions of the properties of the Korea Web graph. We report four power law distributions in the Korea Web. We construct the Korea Web graph, using the SCC (strongly connected component) enumerating algorithm [3]. We also analyze the similarities and differences between the global Web graph and the Korea Web graph.

One contributions of this paper is to show that the Korea Web graph, a subset of the global Web graph, shows different and similar properties in comparison with the global Web graph in terms of the structure of the Web. The research on Korea Web graph is useful for understanding the evolution of Korea Web graph, predicting the growing pace of Korea Web, improving the performance of the Korea Web search engine, and processing the Korea Web information.

## 2 Graph Structure of the Korea Web

For our experiment, we crawled Korea Web pages using our crawler [14] with three machines from June 2006 to July 2006. The IP address is used to determine to see if it belongs to the Korea sites. The crawler downloaded Korea web pages up to the ninth depth, and it collected at most 640,000 web pages from a single site. In order to reduce the size, we compressed links as we downloaded them. The size of the raw data was approximately 65GB, which contain various information on URLs for pages themselves and hyperlinks among pages. We processed the raw data to create Korea Web graph, such as removing invalid URLs, and assigning a unique ID to each URL. We finally constructed the Korea Web graph, which contains 116 million pages (nodes) and 2.7 billion links (edges).

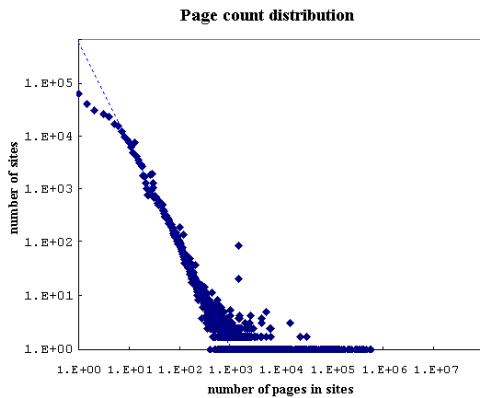
### 2.1 Power Law Distributions

We now consider various power law phenomenon of the Korea Web graph.

**Page Number Distribution in Web Sites.** Figure 1 shows the distribution of page numbers in web sites. The  $x$ -axis shows the number of pages in each site while the  $y$ -axis shows the number of sites which have the corresponding  $x$  pages. Each point  $(x, y)$  on the distribution indicates that  $y$  number of sites have  $x$  pages. This graph exhibits that the distribution of the number of pages in web sites follows the power law while the exponent is roughly 2.3. Under this distribution, the top 18% of sites

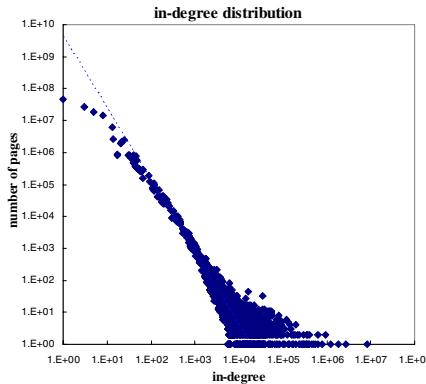
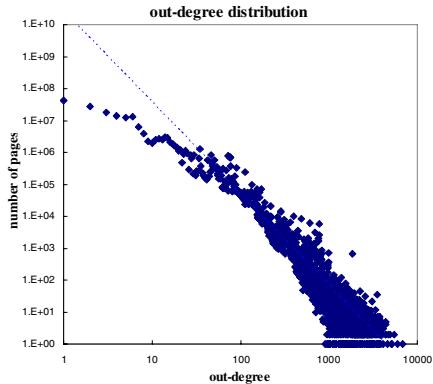
possess about 90% of the total pages of the Korea Web, while 82% of sites contain only 10% of the total pages. This implies that the distribution of the number of pages in web sites of the Korea Web also obeys Pareto's Law (also known as 20:80 law) although the proportion is a little different. The anomalous points at 1417 and 1418 on the x-axis are due to a cluster of sites that have identical web pages even though they have different host names. The site with most pages has about 590 thousand pages.

The Korea Web is different from the China Web which has exponent of 1.74. What it means that the probability that we find web sites in Korea becomes more exponentially decreased than we find such web sites in China, as the number of web pages in a site increases.

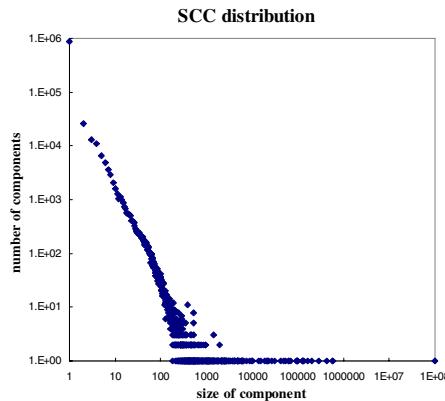


**Fig. 1.** Page number distribution in web sites

**Degree Distribution of the Korea Web.** The degree distributions in the Korea Web also follow the power law distribution. Figure 2 and 3 are a log-log plot of the in-degree and out-degree distributions of the Korea Web graph, respectively. In all our log-log plots, straight lines are linear regressions for the best power law fit. Figure 2 shows that the distribution of in-degree exhibits a power law with exponent roughly 2.2, which is almost the same value as in the global Web [2] and the China Web [4]. Applying an inverse polynomial to the data, we can find the probability that a page has  $i$  in-links to be roughly proportional to  $i^{-2.2}$ . The page that has the most in-links contains as many as 47 million in-links. The out-degree distribution also exhibits a power law while the exponent is roughly 2.8, as in Figure 3. The average out-links in a page of the Korea Web is 27.4. This number is about 3 times more than the average out-links (i.e. eight) which was reported in 1999 [2]. Here we would like to conjecture that the number of links in a page is increasing (at the same time the connectivity among web pages is growing) as time goes by. Note that the pages with out-degrees less than 100 on the x-axis significantly deviate from the best power law fit, suggesting that we might need to have a new distribution to model pages with low out-degrees.

**Fig. 2.** In-degree distribution**Fig. 3.** Out-degree distribution

**Distribution of Strongly Connected Components.** By running the split-merge algorithm [3], we find that there is a single large SCC consisting of about 99 million pages, and all other components are significantly smaller in size. The single large SCC has barely 86% of all the Korean pages. Figure 4 indicates that the distribution of the SCC sizes of the Korea Web also follows a power law with exponent roughly 2.3.

**Fig. 4.** Distribution of strongly connected components

## 2.2 The Macro Structure of the Korea Web

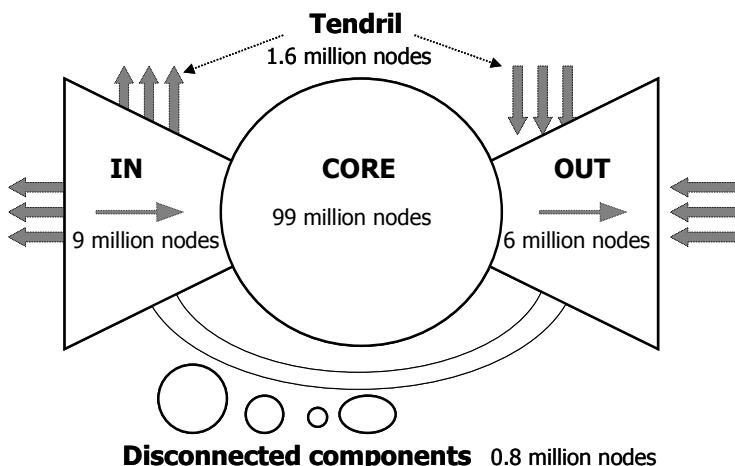
Broder and Kumar [2] present a picture that they refer to a bow-tie of the Web's macroscopic structure. There are four pieces in this structure. The first piece is a central CORE, all of those pages can reach one another along directed links. This giant strongly connected component is at the heart of the Web. The second and third pieces are called IN and OUT. IN consists of pages that can reach the CORE, but cannot be reached from it. OUT consists of pages that are accessible from the CORE,

but do not link back to it. Finally, the TENDRILS contain pages that cannot reach the CORE, and cannot be reached from the CORE.

Figure 5 shows the structure of the Korea Web graph. The Korea Web graph we built contains 116 million pages and 2.7 billion links. The CORE contains about 99 million pages, the IN contains about 9 million pages, the OUT contains about 6 million pages, and the rest contains about 2.4 million pages.

The graph structure of the Korea Web exhibits characteristics that are different to the global Web graph [2] and the China Web graph [4]. The CORE possesses around 86% of the pages of the Korea Web. This is higher than the CORE possessed by the 28% in the global Web graph [2] and 80% in the China Web graph [4]. In other words, the connectivity of the Korea Web is higher than the global and China Web. Furthermore, if pages  $u$  and  $v$  are randomly chosen in the Korea web, the probability that there exists a path between them is at least 0.74 ( $= 86/100 * 86/100$ ), excluding the existence of many tiny SCCs in the Web graph.

The web service providers are currently interested in providing blog and community services to personal users. Personal users tend to create their web pages under a “personalized” frame, which often automatically put links to famous sites in the newly created pages. Business users also like to put links to famous sites in their web pages for various reasons such as increasing accessibility, advertisement and so on. Above all, fast growing are a few giant portal companies, which provide personal services like communities, blogs, thus allow to create a huge number of new pages. The frontrunner companies in Korea would include “www.naver.com”, “www.daum.net” and “www.nate.com”. As the role of such companies become important, more web users get personalized services (private pages are also made in these sites) and companies connect links to popular sites. After all, the Korea web becomes more and more centralized. The observation leads us to expect the size of CORE continue to increase.



**Fig. 5.** The bowtie structure of the Korea Web graph

### 3 Conclusion and Future Work

We studied the Korea Web graph and analyzed the similarities and differences between it and the global and China Web graphs. The CORE of the Korea Web has the bigger portion than the global Web and the China Web do. We learn that the Korea Web is highly connected and centralized. We verified power law distributions in the Korea Web graph from several aspects, and confirmed, as expected, that power law is a basic property of the Web. As for future work, we plan to study about how much does the change of link structure effect the page ranking.

During the experiment, we found many web communities. A Web community is a set of sites that have a similar topic and have many links each other. These communities can be easily found in the form of a single SCC during constructing the Web graph. As a perspective of web search engines, such SCC can be utilized to satisfy given topic-oriented user requests, possibly complementing the existing directory services.

### References

1. Albert, R., Jeong, H., Barabasi, A.: Diameter of the world wide web. *Nature* (1999) 401(6749)
2. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.: Graph structure in the web. the 9th International World-Wide Web Conference (2000)
3. Han, J., Yu, Y., Liu, G., Xue, G.: An Algorithm for Enumerating SCCs in Web Graph. the 7<sup>th</sup> Asia Pacific Web Conference (2005) 655-667
4. Liu, G., Yu, Y., Han, J., Xue, G.: China Web Graph Measurements and Evolutions. the 7<sup>th</sup> Asia Pacific Web Conference (2005) 668-679
5. Cho, J., Roy, S.: Impact of search engines on page popularity. the 13<sup>th</sup> World-Wide Web Conference, (2004)
6. Boldi, P., Codenotti, B., Santini, M., Vigna, S.: Structural properties of the African web. (2002)
7. Ntoulas, A., Cho, J., Olston, C.: What's new on the web? The evolution of the web from a search engine perspective. the 13<sup>th</sup> International World-Wide Web Conference (2004)
8. Kumar, R., Raghavan, P., Rajagopalan, S., Sivakumar, D., Tomkins, A., Upfal, E.: The Web as a graph. *Lecture Notes in Computer Science* 1627 (1999)
9. Fetterly, D., Manasse, M., Najork, M., Wiener, J.: A large-scale study of the evolution of Web pages. *Software – Practice and Experience* (SPE) (2004) 213-237
10. Bar-Yossef, Z., Berg, A., Chien, S., Fakcharoenphol, J., Weitz, D.: Approximating aggregate queries about web pages via random walks. the 26<sup>th</sup> VLDB Conference (2000)
11. Heydon, A., Najork, M.: Mercator: A scalable, extensible Web crawler. the 8th International World-Wide Web Conference (1999) 219-229
12. Barabasi, A., Albert, R.: Emergence of scaling in random networks. *Science* (1999) 509-512
13. Kumar, S. R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Trawling emerging cyber-communities automatically. the 8<sup>th</sup> World-Wide Web Conference (1999)
14. Kim, S.J., Lee, S.H.: Implementation of a Web Robot and Statistics on the Korean Web. the 2<sup>nd</sup> International Conference on Human.Society@Internet (2003) 341-350.
15. Moore, A., Murray, B., H.: Sizing the web. Cyveillance, Inc. White Paper. (2000)

# EasyQuerier: A Keyword Based Interface for Web Database Integration System

Xian Li<sup>1</sup>, Weiyi Meng<sup>2</sup>, and Xiaofeng Meng<sup>1</sup>

<sup>1</sup> School of Information, Renmin University of China

{xianli, xfmeng}@ruc.edu.cn

<sup>2</sup> Computer Science Dept., SUNY at Binghamton

meng@cs.binghamton.edu

**Abstract.** Recently a lot of work on integrating the search interfaces of multiple Web databases of the same domain into an integrated interface has been reported. Such integrated interfaces enable users to search multiple Web databases using one query. However, there are two potential problems when using these integrated interfaces in practice. First, if the number of domains is large, it may be difficult for users to find the correct domain. Second, the integrated interfaces can become too complicated for ordinary users to use. In this paper, we propose a system called EasyQuerier to tackle these problems. EasyQuerier allows the users to submit keyword-based queries to access the Web databases by first mapping a keyword-based user query to a suitable domain and then translating the user query to a well-formatted query on the integrated interface of the found domain. Our experiments show that both our domain mapping and query translation techniques work very well.

## 1 Introduction

A large proportion of the information on the Web is stored in the Web accessible databases [1] which are often called *Web Databases* (WDBs). WDB integration is an emerging technique for providing users an unified way to access multiple WDBs. One key research issue here is to automatically integrate the local query interfaces of the WDBs in the same domain into an integrated query interface [2] [3] [4]. Although this issue has received a lot of attention in recent years, using such integrated interfaces in practice has several problems:

1. One integrated interface is able to access only one specific domain. The users need to first determine the desired domain and then find the corresponding integrated interface to submit queries. As the number of domains grows, domain searching becomes an obstacle for the wide use of the integrated interfaces.
2. The integrated query interfaces can be too complex to use for ordinary users because they typically contain a large number of attributes and many of them have lots of pre-defined values.
3. Each attribute in the integrated interface can accept only one value at a time. So a user has to submit multiple queries when he/she wants to set optional search conditions. For example, if a user wants to search a job with job title “DBA” or “Software engineer”, the user has to submit two queries to the integrated interface.

In this paper we propose a novel solution to overcome the above problems while still supporting unified access to multiple WDBs. Our solution provides a simple keyword-based interface “EasyQuerier” plus two mappings, one maps a user query to the correct domain and the other maps the query to one or more queries on the integrated query interface of the domain. EasyQuerier allows a user to submit queries against any domain. Besides, multiple values corresponding to the same attribute on an integrated interface can be entered in the same query. For the job-hunting example given previously, the user can simply enter “DBA or Software engineer”.

The rest of this paper is organized as follows. Section 2 provides an overview of EasyQuerier. Section 3 describes our domain mapping solution. Section 4 proposes the query translation algorithm from the keyword-based interface to integrated interfaces. Section 5 reports the experimental results and the analysis. Section 6 reviews related work followed with the conclusion in Section 7.

## 2 Overview of EasyQuerier

With EasyQuerier, users only need to provide keyword-like queries. Based on the submitted query, the related domain is determined first; then the query is translated into one or more queries that fit the integrated interface of the selected domain; finally each translated query is mapped to the query interfaces of the local Web databases of the domain. In this paper, we focus only on the first two steps of the above process.

In this paper, we assume that an integrated query interface for each domain has already been constructed using some existing techniques (e.g., the WISE-Integrator [3] [5]). EasyQuerier is built on top of these integrated query interfaces. Users can generally submit keyword queries as what they usually do when querying search engines.

**Example 1.** For the following user query:

**Q1:** New York or Washington, education, \$2000-\$3000  
 three keyword units, {New York, Washington}, {education}, and {\$2000-\$3000} (a range) are obtained and their data types are text, text, and money, respectively.

## 3 Domain Mapping

We aim to map a user query to the correct domain automatically without domain information to be separately entered. We first present a model to represent each domain.

### 3.1 Domain Representation Model

Our survey covering nine different domains shows that near 90% of the attributes have converging value sets. We use the converged value sets to represent each domain. We propose a domain representation model as follows. Specifically, each domain D is modelled by a quadruplet:  $D = \langle d\_ID, CT, AT, VT \rangle$ , where

1.  $d\_ID$  is the unique domain identifier.
2.  $CT = \{ct_i | i = 1, 2, \dots\}$  is a set of Conceptual Terms, which describe the whole domain concept, such as “car”, “vehicles”, “book”, “music CD”.

3.  $AT = \bigcup_{A \in D} DAL(d\_ID, A_i)$  is a set of **Attribute Label Terms** consisting of attribute labels of the products in this domain.  $DAL(d\_ID, A_i)$ , Domain Attribute Label set, is a set of all the terms related to the attribute label of  $A_i$  in domain  $d\_ID$ .  $DAL(d\_ID, A_i)$  consists of terms from three classes: (1)InteLabel: The global label for  $A_i$  in the integrated query interface. (2)LocalLabel: All the labels representing  $A_i$  in the local query interfaces. (3)OtherLabel: It contains some synonyms and immediate hypernyms/hyponyms of those terms in InteLabel and LocalLabel obtained using WordNet.
4.  $VT = \bigcup_{A \in D} DAV(d\_ID, A_i)$ , is a set of the **Value Terms** associated with the products' attributes in the domain  $d\_ID$ .  $DAV(d\_ID, A_i)$ , Domain Attribute Value set, is a set of all the pre-defined values associated with  $A_i$  in domain  $d\_ID$ .  
For **Character Attribute**, values are classified just like for DAL, i.e., we have InteValue, LocalValue, OtherValue. For **Non-text Attribute**, DAV can be characterized by the pre-defined ranges available on the integrated interfaces.

### 3.2 Term Weight Assignment

Often different terms have different ability to differentiate the domains. For example, intuitively attribute label “price” is less powerful than “title” in differentiating the book domain from other domains because the former appears in more domains than the latter. Therefore, we should assign a weight to each term in each domain representation to reflect its ability in differentiating the domain from other domains.

There are different ways to assign weights to a term. In this paper, we adopt a method from [6] that was used in the context of differentiating different component search engines (document databases) in a metasearch. In [6], a statistic called CVV (cue validity variance) is used to measure the skew of the distribution of terms across all document databases, each of which contains a number of documents. For our problem, each domain can be considered as a document database and each local query interface in the domain as a document. Then the CVV of a term can be used as its weight in its ability to differentiate different domains. Denote  $if_{ij}$  as the *interface frequency* of term  $t_j$  in the  $i$ -th domain  $D_i$ , i.e., it is the number of times  $t_j$  appears in either AT or VT in  $D_i$ . Denote  $CVV_j$  as the CVV for  $t_j$ . Then the weight of  $t_j$  in  $D_i$  can be computed by:  $Weight(D_i, t_j) = CVV_j * if_{ij}$ .

### 3.3 Domain Mapping

After the representation of each domain is generated, we can map each query to a certain domain by computing the similarity between the query and each domain.

We now discuss how to compute the similarity between Q and each domain D. As mentioned in Section 2, we parse a query Q into a set of keyword units  $Q = \{u_1, u_2, \dots, u_n\}$ . Therefore, we first compute the similarity between each  $u_i$  and the domain D. Each  $u_i$  may contain one or more query terms denoted as  $\{v_i^1, v_i^2, \dots\}$ . For each  $v_i^x$ , we first calculate its similarity with the best matching term in the representation of domain D. Only terms of the attributes that have compatible data types with the data type of  $u_i$  are considered. Let  $T_i^x$  denote this term set. First, consider the case when  $v_i^x$  is a text type query term. The similarity between  $v_i^x$  and a term  $t_j$  in  $T_i^x$  is

computed by  $\text{Sim}(v_i^x, t_j) = \frac{cw}{\max(|v_i^x|, |t_j|)}$ , where  $cw$  is the number of common words between  $v_i^x$  and  $t_j$ . Now we consider the case when  $v_i^x$  is of a non-text type. In this case,  $\text{Sim}(v_i^x, t_j)$  is computed based on the percentage of  $v_i^x$  that is covered by  $t_j$ , i.e.,  $\text{Sim}(v_i^x, t_j) = \frac{|cr|}{|v_i^x|}$ , where  $cr$  is the shared range between  $v_i^x$  and  $t_j$ . For both cases, we call the term most similar to  $v_i^x$  as  $v_i^x$ 's *matching term* and denote it as  $t_i^x$ .

We now define the similarity between  $u_i$  and  $D$ , denoted  $\text{Sim}(u_i, D)$ , to be  $\max_x \{\text{Sim}(v_i^x, t_i^x)\}$ . Let  $t_i^y$  be the term such that  $\max_x \{\text{Sim}(v_i^x, t_i^x)\} = \text{Sim}(v_i^y, t_i^y)$ . If more than one such  $t_i^y$  exist, take the one with the largest  $\text{Weight}(D, t_i^y)$ . Finally, the similarity between  $Q$  and  $D$  (called the *mapping degree*) is defined as a weighted sum of all the similarities between all the keyword units in  $Q$  and  $D$ , i.e.,

$$\text{Sim}(Q, D) = \sum_{i=1}^n \text{Sim}(u_i, D) * \text{Weight}(D, t_i^y)$$

## 4 Query Translation

Each query has been parsed into several keyword units before domain mapping. The main challenge in query translation is to map each keyword unit to its most appropriate attribute on the integrated interface of the selected domain. In this section, we first introduce a computation model for query translation, later we discuss how to generate query translation solution based on this model.

### 4.1 Computation Model of the Query Translation

**Definition 4.1.** (Keyword-Attribute Matching (KAM)). Given a keyword unit  $u$  and an attribute  $A$  from the integrated interface, their mapping is denoted as  $KAM(u, A)$ .

**Definition 4.2.** (Degree of Matching (DM)). DM is the degree of matching for a KAM, with value range  $[0, 1]$ . Given  $k$  keyword units and  $m$  attributes,  $k * m$  KAMs can be generated and their DM values form a  $k * m$  matrix, which will be called the DM matrix.

**Definition 4.3.** (Query Translation Solution (QTS)). A QTS represents a strategy of filling in the query interface. A QTS is comprised of  $k$  KAMs, where  $k$  is the number of keyword units.

**Definition 4.4.** (Conviction). This measurement determines whether a QTS is reasonable. The larger the DM of a KAM, the more reasonable the KAM is. Thus, the QTS containing such a KAM will more likely yield sounder query translation. Thus the value of Conviction is computed as a weighted sum of all the related DMs.

### 4.2 Computation of DM

In our system,  $DM(u_i, A)$  is determined by the similarity between the keyword unit  $u_i$  and the value set of attribute  $A$ . The value set of  $A$  on the integrated interface of domain  $d\_ID$  is  $DAV(d\_ID, A)$  (see Section 3.1).

A keyword unit in EasyQuerier may contain more than one keyword related to the same attribute. Let  $u_i = \{v_i^1 \text{ or } v_i^2 \text{ or } \dots \text{ or } v_i^p\}$  be such a keyword unit. When computing the DM of a  $KAM(u_i, A_j)$ , we first calculate  $\text{Sim}(v_i^x, A_j)$  which represents the similarity between a value  $v_i^x$  and an attribute  $A_j$ , then the maximum of all the

similarities is the value of  $DM(u_i, A_j)$ . For each  $t_j$  in the DAV of  $A_j$   $Sim(v_i^x, t_j)$  is computed as what mentioned in section 3.3.  $Sim(v_i^x, A_j)$  is the maximum value of all the  $Sim(v_i^x, t_j)$ .

Finally, the  $DM(u_i, A_j)$  is aggregated from all the  $Sim$  values related to the keywords in  $u_i$  using  $DM(u_i, A_j) = \max_{x=1}^p \{Sim(v_i^x, A_j)\}$ .

#### 4.3 Computation of Conviction and QTS Generation

In Definition 4.4, the Conviction value of a QTS is a weighted sum of the DMs of the related KAMs. We compute a weight  $w(A_j)$  for each attribute  $A_j$  based on its *interface frequency*. Let  $if_i$  be the number of local query interfaces that contain attribute  $A_i$ . Intuitively, if an attribute appears in more local interfaces of a domain, it is more important in the domain. Based on this, we compute  $w(A_j) = if_j / (\sum_i if_i)$ . Finally, for  $QTS = KAM(u_1, A'_1) \wedge KAM(u_2, A'_2) \wedge \dots \wedge KAM(u_k, A'_k)$ , we use the following formula to compute its conviction:

$$Conviction(QTS) = \sum_{i=1}^k w(A'_i) * DM(u_i, A'_i)$$

### 5 Experiments

A prototype of EasyQuerier has been implemented. The data collection for the experiment includes: web databases and user queries. (1) Web databases: WDBs covering 9 different domains are collected with 50 databases for each domain. (2) User query collection: 10 students across five different majors are invited as the evaluators of our demo system. For each domain, every student provides two different keyword queries.

The evaluation for both domain mapping and query translation is similar: we identify a *correct mapping/translation* by checking whether the selected domain/translated query with the largest similarity matches the user's intention. If the user is not satisfied with the top result, we let them click the button "more" for more choices. In general, the top 3 choices are provided. If the correct result appears in these choices, we consider the result an *acceptable mapping/translation*; otherwise the mapping/translation is considered to be *wrong*.

**Results on domain mapping.** The experiment on domain mapping is conducted on the 9 domains. For each query, the produced domains are ranked in descending order of their similarities with the query.

Figure 1 shows the overall percentages of the mapping results that are correct, acceptable and wrong, respectively, for all queries as well as for each group of queries. As it can be seen, the overall accuracy is very good. Failures are mostly caused by inadequate information in user queries.

**Results on query translation.** After translating the source query, one or more translated queries are generated. Figure 2 shows the percentages of the translations that are correct, acceptable and wrong for each domain. We find that for the nine domains considered, most queries can be translated correctly. However, for the book, music and movie domains, the average accuracy is lower at about 82.5%. The main cause of failures for these domains is that many important attributes such as "title", "author", "singer", and "director" are textboxes for which building a value set is difficult.

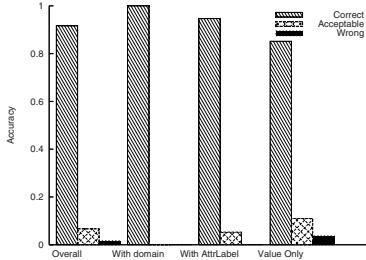


Fig. 1. Domain mapping accuracy

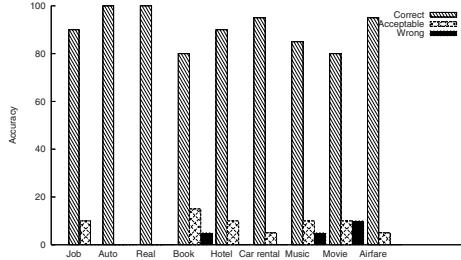


Fig. 2. Query translation accuracy

## 6 Related Work

Automatic interface integration has been a hot issue in recent years. WISE-integrator [3] and Meta-Querier [2] aim at integrating the complex query interfaces provided by WDBs. As discussed in Section 1 these integrated query interfaces are likely to be too complex for ordinary users and our work aims to provide an easy-to-use interface.

Our work is related to researches that translate natural language queries to structured queries (such as SQL) to support natural language access to structured data (e.g., [7][8]). The main differences between these works and our work reported here are as follows. First, they do not deal with the domain mapping problem while we do. Second, they deal with mostly relational databases while we deal with Web query interfaces. Third, they have access to both the schema information and the actual data but we only have access to the schema and very limited pre-defined values available on the query interface but do not have access to the full data. Finally, we deal with keyword queries rather than real natural language queries.

## 7 Conclusion

In this paper, we proposed a novel keyword based interface system EasyQuerier for ordinary users to query structured data in various Web databases. We developed solutions to two technical challenges, one is how to map keyword query to appropriate domains and the other is how to translate the keyword query to a query for the integrated search interface of the domain. Our experimental study involving real users showed that our solutions can produce very promising results.

**Acknowledgment.** This work is supported in part by the NSF of China under grant #s 60573091, 60273018; NSF of Beijing under grant #4073035; Program for New Century Excellent Talents in University (NCET); US NSF grants IIS-0414981 and CNS-0454298.

## References

1. BrightPlanet: The deep web: Surfacing hidden value. (<http://brightplanet.com>)
2. Chang, K.C.C., He, B., Zhang, Z.: Toward large scale integration: Building a metaquerier over databases on the web. In: CIDR. (2005) 44–55

3. He, H., Meng, W., Yu, C.T., Wu, Z.: Wise-integrator: An automatic integrator of web search interfaces for e-commerce. In: VLDB. (2003) 357–368
4. Dragut, E.C., Wu, W., Sistla, A.P., Yu, C.T., Meng, W.: Merging source query interfaces on web databases. In: ICDE. (2006) 46
5. He, H., Meng, W., Yu, C.T., Wu, Z.: Wise-integrator: A system for extracting and integrating complex web search interfaces of the deep web. In: VLDB. (2005) 1314–1317
6. Yuwono, B., Lee, D.L.: Search and ranking algorithms for locating resources on the world wide web. In: ICDE. (1996) 164–171
7. Androultsopoulos, I., Ritchie, G.D., Thanisch, P.: Natural language interfaces to databases - an introduction. CoRR **cmp-lg/9503016** (1995)
8. A. Popescu, O.E., Kautz, H.: Towords a theory of natural language interfaces to databases. International Conference on Intelligent User Interfaces. (2003)

# Anomalies Detection in Mobile Network Management Data

Marco Anisetti<sup>1</sup>, Claudio A. Ardagna<sup>1</sup>, Valerio Bellandi<sup>1</sup>, Elisa Bernardoni<sup>1</sup>, Ernesto Damiani<sup>1</sup>, and Salvatore Reale<sup>2</sup>

<sup>1</sup> Department of Information Technology, University of Milan  
via Bramante, 65 - 26013, Crema (CR), Italy

{damiani, anisetti, ardagna, bellandi, bernardoni}@dti.unimi.it  
<sup>2</sup> Siemens S.p.A.

Carrier Research & Development Radio Access - Network Management  
Via Monfalcone 1, 20092  
Cinisello Balsamo (MI), Italy  
salvatore.reale@siemens.com

**Abstract.** Third generation (3G) mobile networks rely on distributed architectures where Operation and Maintenance Centers handle a large amount of information about network behavior. Such data can be processed to extract higher-level knowledge, useful for network management and optimization. In this paper we apply reduction techniques, such as *Principal Component Analysis*, to identify orthogonal subspaces representing the more interesting data contributing to overall variance and to split them up in “normal” and “anomalous” subspaces. Patterns within anomalous subspaces allow for early detection of network anomalies, improving mobile networks management and reducing the risk of malfunctioning.

## 1 Introduction

Third generation (3G) mobile networks must satisfy demanding performance requirements. Switching nodes, called *Network Elements* (NE), handle many more calls at a time than in earlier networks. NE monitoring is carried out by *Operation and Maintenance Centers* (OMC), dealing with a huge amount of data. Mobile network data are profoundly different from data collected in traditional IP traffic analysis. On IP networks, in fact, years of research have produced knowledge bases supporting association between data anomalies and their semantics. In the mobile environment, normal network behavior is usually represented through a set of templates, and traffic data analysis is still at an early stage. In this paper, we discuss the problem of extracting semantic information from mobile network management data. We focus on anomaly detection, providing a method to find out where network behavior deviates from the normal one. Our data space is represented by a three-dimensional matrix (*NE, Counters, Time*) where counters are homogeneous entities, and NEs are supposed to have similar behavior, because they are chosen in a geographically limited area. We use data reduction techniques such as *Multiway Principal Component Analysis* (MPCA) to map the huge space representing the whole dataset into a reduced subspace which is, then, split

into two parts creating a *normal* and an *anomalous* subspace, whose axes are orthogonal, i.e. all variables are independent. Our reduction technique, specifically designed to exploit the properties of mobile network data, can be seen as “short-term data mining”, because analysis is carried out off-line, but without the need for huge historic databases.

## 2 Related Work

To the best of our knowledge, no work has been published on mobile network traffic anomalies detection and identification. Quite a number of research papers have dealt with the problem of representing and processing IP network traffic measurements. These two areas, though related, are however distinct; intuitively, connections on a packet-switching network like the Internet are characterized by a more diverse set of parameters than calls on a circuit-switching network where most performance data is relative to call setup<sup>1</sup>. The seminal work [6], which presented the first large-scale analysis of flow traffic in IP networks, decomposes the structure of flow time series into three main constituents: common periodic trends, short-lived bursts, and random noise. Each traffic type brings into focus a different set of anomalies spanning a remarkably wide spectrum of semantically recognizable event types, including denial of service attacks (single-source and distributed), flash crowds, port scanning, downstream traffic engineering, high-rate flows, worm propagation, and network outage. *Principal Components Analysis* reduction techniques have been successfully adopted in several research fields such as IP traffic analysis and industrial processes monitoring. Specifically, [9] investigates the suitability of using optical emission spectroscopy (OES) for fault detection and classification of plasma etchers. The paper uses *Multiway Principal Component Analysis* (MPCA) to assess the sensitivity of multiple scans within a wafer with respect to typical faults. MPCA has also been successfully applied to monitoring chemical batch and semi-batch processes. For instance in [5] a new method combining *Independent Component Analysis* (ICA) and MPCA is proposed. ICA is used to express independent variables as linear combinations of MPCA *latent variables*.

After data reduction, mathematical tools (like wavelet [2]) for signal analysis are used to achieve network traffic characterization. Finally, some recent work deals with compressing information collected in a sensor network environment. Deligiannakis et al. [3] presented a technique for compressing multiple streams of sensor data, exploiting correlation among multiple measurements on the same sensor. However, while sensor data analysis is somewhat related to network data monitoring, the statistic properties of the data streams turn out to be very different [3].

## 3 Mobile Network Data

Mobile *Radio Access Network* monitoring is carried out by *Operation and Maintenance Centers* (OMC). Each OMC can manage hundreds of *Base Station Controllers* (BSCs) that control the lower level of the network hierarchy that is composed by thousands

---

<sup>1</sup> Once a call is established, a fixed bandwidth is allocated to its bit flow for its whole duration, so no further analysis of time-variant behavior of data delivery is necessary.

**Table 1.** Sample counters (mobile network) with 1 hours bin

|                | NE-001 | NE-002 | NE-003 | NE-004 | NE-005 | NE-006 | NE-007 | NE-008 | NE-009 |
|----------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 22/12/05 06.00 | 26,11  | 14,4   | 14,75  | 19,30  | 21,95  | 21,93  | 14,99  | 10,70  | 16,22  |
| 22/12/05 07.00 | 34,41  | 14,64  | 18,11  | 20,41  | 32,04  | 25,14  | 17,62  | 12,75  | 17,13  |
| :              | :      | :      | :      | :      | :      | :      | :      | :      | :      |
| 23/12/05 04.00 | 12,33  | 11,24  | 11,33  | 10,43  | 13,1   | 22,02  | 10,67  | 12,98  | 12,67  |
| 23/12/05 05.00 | 13,62  | 11,97  | 18,29  | 10,37  | 12,58  | 21,55  | 10,75  | 14,20  | 12,26  |

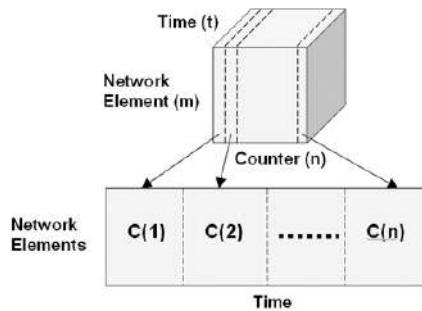
of *Network Elements* (NEs)<sup>2</sup> OMCs periodically collect a number of raw performance data, such as *Physical Link Measurements* (e.g. the bit error rate or the uplink/downlink signal-to-noise ratio), *Message Flow Measurements* (e.g. the packet drop rate) and *Call Measurements* (e.g. call establishments, call drops, total calls, call handover requests, handover failures), produced by BTSs. These raw measurements are called *counters*. Counters collected by OMCs are usually stored into a relational database. Every day, each NE inserts many megabytes of data into the database. In the simplest arrangement (see Table I) this database contains a table for each class of measure<sup>3</sup>, later to be queried and processed to assess network performance. Each column corresponds to a NE counter, while rows correspond to NEs states at a given time. In general, counters show strong autocorrelation, but is very difficult to map counters' semantics to their spectral properties. This is due to *null values* corresponding to missing measurements or events and to *random values* due to unpredictable NE behavior. Also, high correlation between counters of different classes suggests that counters' semantics, even when guessed correctly, cannot be used as a predictor of their spectral properties. This behavior is markedly different from the one of Internet traffic indicators [1]. An anomaly on one node of the IP network propagates to the subsequent nodes in the path, while mobile network anomalies most counters are not subject to linear propagation (with the exception of handover counters). Also, mobile data have an intrinsic locality and therefore can be analyzed in local sub-windows on time domain.

## 4 Semantic Pattern Identification in Management-Related Mobile Data

We now describe our data reduction and analysis techniques. Then, we show an example of anomaly detection over a data set coming from the Italian mobile network. A critical step in mobile data analysis is the selection of a data reduction technique. Available reduction techniques include: *i) Independent Component Analysis (ICA)*, a statistical technique for revealing hidden factors that underlie sets of random measurements[5]; *ii) Principal Component Analysis (PCA)*, a multivariate procedure which rotates data sets so that components giving the largest contribution to data variance are projected

<sup>2</sup> We refrain from giving a full description of 3G mobile network architecture, as this is outside of the scope of the paper.

<sup>3</sup> Each class is composed of different measure characterized by one or more counters.



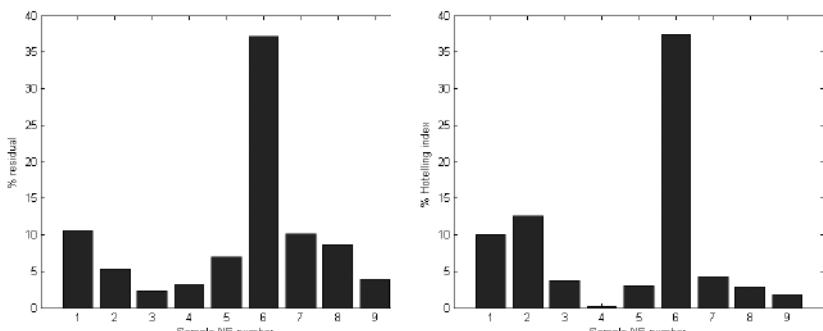
**Fig. 1.** PCA matrix creation process for MPCPA

onto coordinate axes; *iii) Multi-way Principal Component Analysis (MPCA)*, an extension to traditional PCA. It is used to manage  $n$ -dimensional data sets and bring them back to 2-dimensional sets through a *data unfolding* process. MPCA is largely used for analyzing time-variant batch processes [5]; *iv) Relevant Component Analysis (RCA)*, a method that tries to down-scale global data variability. RCA performs a projection of the input data into a feature space via a linear transformation which assigns a large weight to relevant dimensions and small weight to irrelevant ones [8]. PCA is very similar to RCA except for the fact that PCA compresses data along the dimensions that show the smallest variability, while RCA compresses them along the dimension of highest variability. RCA is not suitable to our purposes because we cannot distinguish a-priori between relevant and irrelevant variability. ICA requires more information than just the covariance matrix and is more likely to be used in case of a single physical data source. PCA seems to be the most suitable techniques for our environment, but it cannot be used as it is because we must deal with a multi-dimensional dataset matrix. Therefore, we rely on a MPCA technique [7, 4] applying regular PCA on two-way data sets unfolded from our multi-way data set. Our data come naturally organized in three dimensions: *(i)* Network Element, (NE) *(ii)* counter and *(iii)* (discrete sampling) time. We monitored several counters belonging to different NEs. Our analysis, however, concerns only NEs where the same class of measures is taken (*homogeneous counters*). By applying MPCA to each NE, we extract the history of the evolution of a number of homogeneous counters related to a single measure of interest. As far as data unfolding is concerned, several strategies are available. A first approach, aimed at detecting faults within NEs, arranges all samples and all variables of a NE in a single row. In this way each sample represents a different NE. Figure 1 shows the unfolding of data matrices corresponding to different NEs into a single unfolded data matrix. A second possible approach to unfold the matrix is arranging data so that each row contains a sampling time and each column contains the data of one counter for each NE. Hybridizing these two strategies, one can: *(i)* perform a SPE and  $T^2$  analysis on one counter for every NE; *(ii)* select the most important counter for the entire NE set. In this paper we exploit the first approach. A PCA model is then developed using the entire unfolded data matrix. Matrix  $X$  is defined as  $m \times (n * t)$  mobile data matrix, where  $m$  represents the Mobile Network Elements subject to our measurement,  $n$  represents the size of the selected measure and  $t$  represents the number of bins in which the time series is partitioned. In

this paper,  $t$  is a number of measurement depending on bin size (in minutes). Bin size is a variable of the type of measures which we want to analyze or check and it can vary from 5 to 60 minutes and over, because most anomalies in our datasets lasted less than 5 minutes and showed up as a spike at a single point in time. On the other side,  $n$  can vary depending on the measure we want to analyze. After unfolding, we rely on the assumption that counters belonging to aggregated mobile network elements are highly correlated. We then apply a Principal Component Analysis on the unfolded data matrix  $X$  to reduce data dimensions and to compute two separate subspaces, representing respectively normal and anomalous behavior. The benefits of our approach are threefold. Firstly, the computational overhead of the subsequent processing steps is reduced. Secondly, noise is isolated from the signals. Thirdly, a projection into a subspace of a very low dimension is useful for data visualization.

## 5 Applying Data Reduction and Analysis Techniques to a Real Scenario

We tested our algorithm against a set of mobile network traffic data from the Italian Mobile Network (IMN). IMN is composed of 40 BSC in GSM technology and it has more than 70000 counter trends. About 55000 of these counter trends are zero-mean or null, and the remaining part is composed of linear combination of Gaussian curves, small oscillation, constant, steps, pulses. We tested our algorithm against data gathered from a single BSC for each NE during a ten days period. In this paper we work with data sampled every 15 minutes, i.e. the periodicity usually adopted in mobile network management. Table I contains an example of a data set representing the traffic counters in 9 different, though correlated, network elements. This table represents the normal behavior of the counter. We considered a 3D table with more counters, then applied the PCA technique obtaining a 2D matrix  $X$ . Applying PCA, we also compute a reduced subspace describing principal components. Now, we need to extract only components that contribute most toward explaining NE counters variability. We limit our PCA to 3 components which capture a great percentage of variability ( $> 95\%$ ). This way, we



**Fig. 2.** Anomaly detection with 3 PCA component selected using Multiway PCA. SPE and  $T^2$  index indicate that NE 6 is anomalous.

create the subspace representing the normal behavior of the network. Then, we define a region of acceptable variability for display purposes (*acceptable region*). Our algorithm points out anomalous behavior happening on the network without trying to distinguish real anomalies from false positives. This is mainly due to current lack of a knowledge base mapping mobile network traffic anomalies and their semantics. We relied on human network administrators' expertise to identify real anomalies. Both SPE and  $T^2$  indices show that NE 6 (see Figure 2) exhibits anomalous behavior.

## 6 Conclusion

Anomaly detection is a crucial issue in 3G mobile network data analysis. In this paper we described a promising technique for applying Multiway Principal Component Analysis (MPCA) to mobile network data. Our solution greatly simplifies anomaly search. Our experience with real mobile network traffic datasets suggests that MPCA can be a good correlation as well as anomalies detection technique.

## References

1. M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Trans. Netw.*, 5(6):835–846, 1997.
2. M.E. Crovella and E.D. Kolaczyk. Graph wavelets for spatial traffic analysis. San Francisco, California, April 2003.
3. A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. Paris, France, June 2004.
4. P. Geladi. Analysis of multiway (multi-mode) data. *Chem. Intell. Lab.*, 7:11–30, 1989.
5. N. He, J. Zhang, and S. Wang. Combination of independent component analysis and multi-way principal component analysis for batch process monitoring. *IEEE International Conference on Systems, Man and Cybernetics*, 2004, 1:530–535, October 2004.
6. A. Lakhina, K. Papagiannaki, M.E. Crovella, C. Diot, E.D. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. In *Proc. of ACM SIGMETRICS*, New York, NY, June 2004.
7. K. Esbensen S. Wold, P. Geladi and J. Ohman. Multiway principal components and pls-analysis. *Journal Chemometr.*, 1:41–56, 1987.
8. N. Shental, T. Hertz, D. Weinshall, and M. Pavel. Adjustment learning and relevant component analysis. In *Proc. of European Conference on Computer Vision 2002 (ECCV)*, Copenhagen, Denmark, 2002.
9. H. H. Yue, C. Nauert S. J. Qin, R. J. Markle, and M. Gatto. Fault detection of plasma etchers using optical emission spectra. In *IEEE transaction on semiconductor manufacturing*, volume 13, 2000.

# Security-Conscious XML Indexing

Yan Xiao, Bo Luo, and Dongwon Lee

The Pennsylvania State University, University Park, USA  
xiaoyan515@gmail.com, {bluo, dongwon}@psu.edu

**Abstract.** To support *secure* exchanging and sharing of XML data over the Internet, a myriad of XML access control mechanisms have been proposed. In the setting of node-level fine-grained access control, query evaluation is a process of locating XML nodes that (1) satisfy query constraints, and (2) do not violate security policies. In this regard, we propose and empirically validate a suite of XML indices for multi-level XML security model.

## 1 Introduction

Recently, many proposals focusing on XML security models or enforcement mechanisms have appeared (e.g., XACML, [3], [4]). However, XML query processing issues using indices in dealing with secure XML data has gotten little attention. In this paper, we are interested in devising XML indexing methods to efficiently support Multi-level security model (e.g., [7]) for XML data.

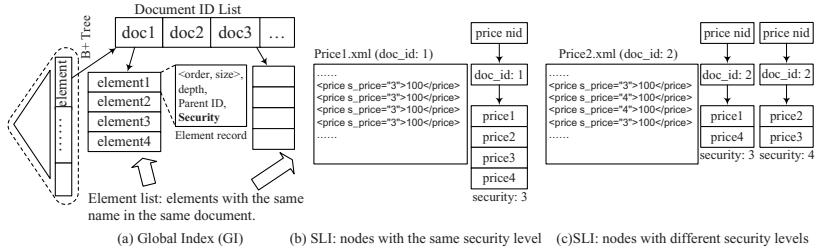
**Motivation:** Imagine a company that has three-level security policy: {Top Secret, Secret, Public}, denoted as {3, 2, 1}, respectively. In the following XML data, each node has an associated security level in the “`s_nodename`” attribute:

```
<Dept s_Dept='1'>
 <Manager s_Manager='1'><Name s_Name='1'>Tom</></><Staff s_Staff='1'><Name s_Name='1'>Jane</></>
 <Proj s_Proj='2' pname='Security' s_pname='2'>
 <Year s_Year='2'>2004</> </Year> <Budget s_Budget='3'>300K</> </Proj>
</Dept>
```

When a user “Tom” with security level ‘2’ issues  $Q://\text{Proj}/\text{Budget}$ , he would not receive budget information for the `pname='Security'` project due to insufficient security level. In this case, enforcing the right access controls of the query  $Q$  by “Tom” is amount to evaluating  $Q'://\text{Proj}[@s_Proj \leq 2] / \text{Budget}[@s_Budget \leq 2]$ . When there are hundreds of such `Proj` and `Budget` elements in documents, therefore, quickly locating those elements with security level  $\leq 2$  plays a critical role in improving the “secure” query processing. The goal of this paper is, therefore, to devise efficient indexing schemes for such a scenario. We use the notations:  $SL(n)$ ,  $SL(q)$  (or  $\{L\} : q$ ) for the security level of node  $n$  and query  $q$ , and  $MinSec(n, D)$  for minimum  $SL(n)$  of nodes in document  $D$ .

## 2 Background and Related Work

Multi-Level Access Control model (e.g. [7]) assigns each object (e.g., node) and each subject (e.g., user) a *security level*, and enforces a rule: “*a level  $L_i$  subject*



**Fig. 1.** Global Index and Single-Level Index

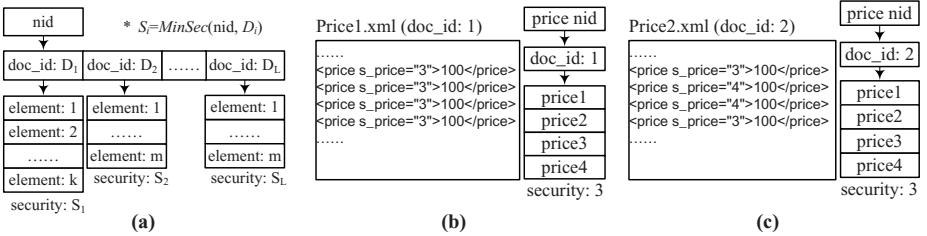
cannot access a level  $L_j$  object unless  $L_i \geq L_j$ ”. Its simplicity enables the wide acceptance in military or organizational applications, but since it requires “total order” among levels, it is less flexible. More flexible models (DAC or RBAC), allow lattice-based levels and are supported by commercial DBMS. In [4], a specific authorization sheet is associated with each XML document/DTD. [3] extends [4] by enriching the supported authorization types, providing a complete description of specification and enforcement mechanism. In [1], an access control environment for XML documents and techniques to deal with authorization priorities and conflict resolution issues are proposed.

We base our approach on XISS [6]. It uses a unique numbering scheme to quickly join ancestor-descendent nodes, and at the bottom layer of the index, information per node is sorted by document order to support fast sort-merge. [2] studies query evaluation methods by exploiting the properties of security model. Our indexing method is complementing [2]. The concept of “two-tier” introduced in [5] (for RDBMS) is the basis of our work. We first adopt two-tier index to XML context, and improve it further.

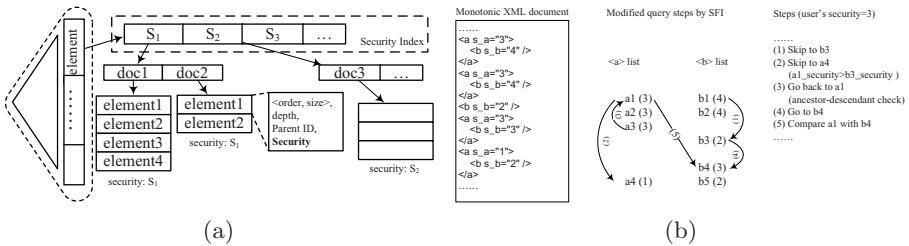
### 3 Security-Conscious XML Index

**Global Index.** As the baseline approach, *Global Index* constructs a regular XML index. Here we choose *element index* of XISS. As shown in Figure 1(a), all element tags are assigned a unique ID, then indexed via  $B^+$  tree at top. The bottom of the tree points to a *document ID list* that contains a list of document IDs where the element appears. Further, each item of the document ID list points to an *element list*, which stores element-related information. To support XML access control, we add additional “security level” of each element in the element list. In this scheme, security check is done at individual element level. Therefore, it could be acceptable for users with high security levels, since they are likely to access most data anyway, but could be inefficient for most users.

**Single-Level Index (SLI).** This approach is to have separate index for each security level. Thus, when a query {2}://Proj is issued, one can simply look for Proj elements from two *single-level* indices: security level 1 and 2. Since all elements in that index are guaranteed to satisfy the specified security constraint,



**Fig. 2.** (a) Modified SLI; (b) and (c) Examples of the modified SLI



**Fig. 3.** (a) Minimum-Security Index (MSI); (b) Special case of SFI

there is no need for additional check. This approach is efficient when elements with the same name have the same security level within an document because the document ID and elements can be stored in one security class. However, if elements have different security levels within a document, all information has to be stored in different security levels. This will cause large storage and query overhead. Figure 2(b) and (c) show examples of both scenarios.

**Minimum-Security Index (MSI).** Both the GI and SLI have pros and cons. To retain only pros of each, we adopt *Two-Tier Coarse Index* [5] (from relational model) and fit it into XML model. Further, we improve it with the *MinSec* concept, making it *Minimum-Security Index (MSI)*.

A problem of GI is that the document IDs we get from  $B^+$  tree may not contain satisfactory IDs at all. Suppose an element *Budget* appears in document  $D_7$  with security level 4. For a query  $q_1, \{3\}://Budget$ , we do not have to retrieve *Budget* from  $D_7$  to check their security levels since  $SL(q_1) = 3 < MinSec(Budget, D_7) = 4$ . If we maintain a reverse link atop single-level index that, for each *MinSec*, points to a document ID list (e.g., *MinSec* 4 points to  $D_7$ ), then significant saving can be made by not visiting unnecessary documents. To use *MinSec*, SLI is modified as shown in Figure 2(a). Here, we keep element list the same as GI, and store document ID into separate security classes based on *MinSec* values. The modified SLI improves SLI by reducing multiple storage and avoiding querying the same document multiple times. The example XML documents in modified SLI are shown in Figure 2(b) and (c).

The MSI, illustrated in Figure 3(a), combines pros of GI and SLI, but exploits the document-level security check by avoiding retrieving unnecessary documents. Therefore, when elements in a document have unique security level (e.g., all `Proj` element in  $D_1$  have security level 4), the MSI is the most advantageous. However, when elements in a document have various security levels, it becomes less efficient. For instance, a document  $D_1$  has 1,000 `Budget` elements, where all have security level 5, except one with 2. Then, for a query  $\{3\}://\text{Budget}$ , even if there is only one element satisfying the security constraint, the MSI still retrieves all 1,000 `Budget` elements since  $\text{MinSec}(\text{Budget}, D_1)$  is 2.

**Skip-Record Index (SRI).** One of the most time-consuming steps in query processing is to retrieve elements from element list and perform sort-merge using ancestor-descendent relationship. To speed up this step, we can avoid sort-merge for those element pairs which cannot satisfy security constraint. One can sort elements by security levels to quickly determine whether or not to continue checking security, but this is not possible since elements in the element list are already sorted by their order (i.e., pre-ordering in the XISS). To solve this, we maintain another number per each element  $e_1$ , called *Skip-Record* that either (1) quickly tells how many element records to skip to get to the next element  $e_2$  that satisfies  $SL(e_2) < SL(e_1)$ , or (2) is “-1” if there is no more such element left in the element list. Consider the following situation:

```

1: <Proj s_Proj='3'> # Skip-Record=2 4: <Proj s_Proj='2'> # Skip-Record=-1
2: <Proj s_Proj='4'> # Skip-Record=1 5: <Proj s_Proj='6'> # Skip-Record=0
3: <Proj s_Proj='5'> # Skip-Record=0 6: <Proj s_Proj='3'> # Skip-Record=-1

```

The Skip-Record value “2” for node 1 implies that one needs to skip “two” elements to get to lower security level. The Skip-Record value “-1” of node 4 suggests that there is no elements with lower security level. For query  $\{1\}://\text{Proj}$ , since the security level of the first `Proj` is 3, it is not satisfactory for the given query. Instead of checking element 2 and 3, we can use the Skip-Record to quickly “skip” two records and go to element 4 directly. When this forth element is not again satisfying the security constraint, one can quit searching in this element list since the Skip-Record of the forth `Proj` is “-1”.

**Skip-Forward Index (SFI).** Consider the following document:

```

<root>
 <a s_a='3'> <b s_b='4'> </> # a1, b1 <a s_a='3'> <b s_b='4'> </> # a2, b2
 <a s_a='3'> <b s_b='4'> </> # a3, b3 <a s_a='3'> <b s_b='4'> </> # a4, b4
 <a s_a='1'> <b s_b='2'> </> # a5, b5
</root>

```

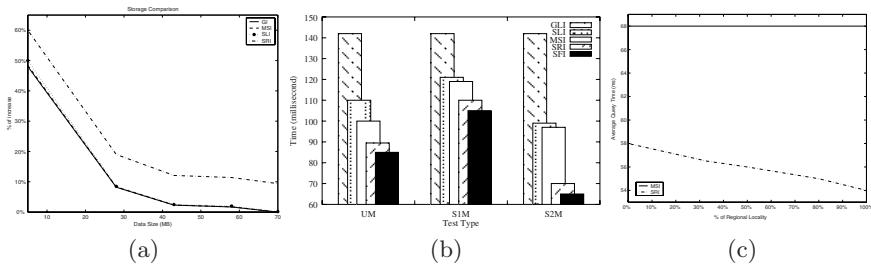
In processing query “`a/b`”, sort-merge between two lists are needed. Furthermore, the following depicts two such lists with (SL, Skip-Record) for each element.

```

a-list -- a1(3,3) a2(3,2) a3(3,1) a4(3,0) a5(1,-1) ...
b-list -- b1(4,3) b2(4,2) b3(4,1) b4(4,0) b5(2,-1) ...

```

Consider query  $\{3\}://\text{a}/\text{b}$ . `a1-b1` pair is first compared, since it satisfies ancestor-descendent relationship, its security is checked; `a1`’s security is satisfied, but `b1`’s is not, thus `b1` is not returned. At this point, according to `b1`’s Skip-Record, we can skip the next “3” `<b>`, and examine `b5` immediately. However, for the `a`-list side,



**Fig. 4.** (a) Index size comparison; (b)-(c) Query evaluation for monotonic model

next record to be evaluated is  $a_2$  since  $a_1$  was satisfied.  $a_2$  again satisfies security constraint, but  $a_2-b_5$  fails ancestor-descendent relationship check. Then we have to continue checking all remaining  $\langle a \rangle$  until it reaches  $a_5-b_5$ , which satisfies both ancestor-descendent relationship and security constraint (i.e.,  $\leq 3$ ).

However, in the “monotonic” security model where ancestor’s security levels are guaranteed to be not higher than descendants’, we can avoid security check of ancestors. In the above example, after  $a_1-b_1$  pair is examined and the next element to examine is determined to be  $b_5$ , according to the Skip-Record number of  $b_1$ , we can immediately rule out  $a_2-a_4$  since their security levels are higher than  $b_5$ ’s and thus none of them can be ancestor of  $b_5$  in the monotonic model. In general, in the monotonic security model, if  $a$ ’s SL is greater than  $b$ ’s SL, then using the Skip-Record numbers, we can skip to the next  $a$  whose SL is no greater than  $b$ ’s SL. If current  $a$ ’s Skip-Record number is “-1”, then we move to the next  $a$  and  $b$ , and continue the above steps. If  $a$ ’s SL is not greater than  $b$ ’s, we can then check ancestor-descendant relationship. There is a special case as shown in Figure 3(b):  $b_3$  has no  $a$  ancestor. If we use above algorithm, we jump to  $a_4$  after find  $a_1$ ’s security value is greater than  $b_3$ ’s. So, we miss  $a_3-b_4$  because we skipped  $a_3$ . Therefore, after we find that  $a_4$  is after  $b_3$  and is not its ancestor, we need to go back and check with  $b_4$ .

## 4 Experimental Validation

We implemented the five variations in the XISS system for evaluation. We have generated three variations of security distribution for monotonic data: uniform distribution for (UM), skewed security distribution with more low security level data (S1M), skewed security distribution with more high security level data (S2M); and the same for non-monotonic data: UNM, S1NM and S2NM.

**Index Size:** The index sizes are compared in Figure 4(a), where % of the increase of index size was measured for different XML sizes. SRI incurs the most index space increase since it maintains Skip-Record for each item in element list, but the additional storage overhead is not substantial. When data size is small, the huge increase is due to the default size allocation by XISS. As data size increases, the overhead is within 10% range and almost 1% for other methods.

**Query Evaluation Time:** The impact of security model selection (monotonic or non-monotonic) towards system performance is minor. Figure 4(b) shows the result of five variations over three data sets under monotonic model, and the case for non-monotonic is similar. Regardless of the skewness of the security information, SRI and SFI outperforms conventional ones significantly.

SRI is more efficient when elements in the list are sorted by security values in ascending order. We test query performance on different regional locality in a document. First, we define a *block* for element  $E$  in an XML document as the region where all security levels of  $E$  elements are the same or sorted by ascending order. Then, suppose in document  $D$ ,  $E$  appears  $n$  times in  $b$  blocks. The regional locality of  $E$  in  $D$  is calculated as:  $R(E, D) = 100\%$ , when  $b = 1$ ;  $R(E, D) = (n - b)/n$ , otherwise. As shown in is shown in Figure 4(c), SRI is more efficient when  $R(E, D)$  increases. This is because the whole block of records is skipped if the first element can not satisfy security check.

## 5 Conclusion and Future Work

In this paper, we consider five index schemes that support multi-level XML access control – Global Index, Single-level Index, Minimum Security Index, Skip-Record Index, and Skip-forward Index. By utilizing the characteristics of XML model and monotonic/non-monotonic security models, SRI or SFI improves other variations up to 130% at best. In general, all the proposed indices can effectively take advantage of pre-security checks, while not intruding the original XML database like XISS and their path join algorithms. Thus, our extension is quite practical.

## References

1. E. Bertino and E. Ferrari. “Secure and Selective Dissemination of XML Documents”. *IEEE Trans. on Information and System Security*, 5(3):290–331, Aug. 2002.
2. S. Cho, S. Amer-Yahia, L. V.S. Lakshmanan, and D. Srivastava. “Optimizing the Secure Evaluation of Twig Queries”. In *VLDB*, Hong Kong, China, Aug. 2002.
3. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. “A Fine-Grained Access Control System for XML Documents”. *IEEE Trans. on Information and System Security*, 5(2):169–202, May 2002.
4. E. Damiani, S. De Capitani Di Vimercati, S. Paraboschi, and P. Samarati. “Design and Implementation of an Access Control Processor for XML Documents”. *Computer Networks*, 33(6):59–75, 2000.
5. S. Jajodia, R. Mukkamala, and I. Ray. “A Two-tier Coarse Indexing Scheme for MLS Database Systems”. In *IFIP WG 11.3 Working Conf. on Data and Applications Security (DBSec)*, Lake Tahoe, CA, Aug. 1998.
6. Q. Li and B. Moon. “Indexing and Querying XML Data for Regular Path Expressions”. In *VLDB*, Roma, Italy, Sep. 2001.
7. S. Osborn. “Mandatory Access Control and Role-Based Access Control Revisited”. In *ACM Workshop on Role Based Access Control*, pages 31–40, Fairfax, VA, 1997.

# Framework for Extending RFID Events with Business Rule\*,\*\*

Mikyeong Moon<sup>1</sup>, Seongjin Kim<sup>1</sup>, Keunhyuk Yeom<sup>1</sup>, and Heeseok Choi<sup>2</sup>

<sup>1</sup> Department of Computer Engineering, Pusan National University

30 Jangjeon Dong, Geumjeong Ku, Busan, 609-735, Korea

{mkmoon, sj179, yeom}@pusan.ac.kr

<sup>2</sup> NTIS Organization, Korea Institute of Science and Technology Information

Eoeun-dong 52-11, Yuseong-gu, Daejeon, 305-806, Korea

choihs@kisti.re.kr

**Abstract.** Radio frequency identification (RFID) technology is believed to be the next revolutionary step in supply-chain management. Complex process simplification using RFID technology can offer particularly important benefits to many enterprises. To derive real benefit from RFID, the application must rapidly implement functions to process the large quantity of event data generated by RFID operations. For this reason, developers are forced to implement systems to derive meaningful high-level events from simple RFID events. Although applications could directly consume and act on RFID event, extracting the business rules from the business logic leads to better decoupling of the system, which consequentially, increases maintainability. In this paper, we describe an RFID business aware framework for extending RFID events using business rules, and then processing these to show complex events.

**Keywords:** RFID, RFID event, business rule, RFID business event, RFID application development framework.

## 1 Introduction

RFID may dramatically change an organization's capability to obtain real-time information of the location and properties of tagged people or objects. To derive real benefit from RFID, the application must implement functions to rapidly process the large quantity of event data generated by the RFID operations. Recently, many RFID middleware systems have been developed by major corporations [1, 2, 3, 4]. Although RFID middleware deletes duplicate readings from the same tag and helps manage the flow of data, developers are required to implement systems to derive meaningful high-level events, which contain more useful knowledge for the application than the simple RFID events. The application developer must collect RFID events, access the

---

\* This work was supported by the Brain Korea 21 Project in 2007.

\*\* This work was supported by the Regional Research Centers Program (Research Center for Logistics Information Technology), granted by the Korean Ministry of Education & Human Resources Development.

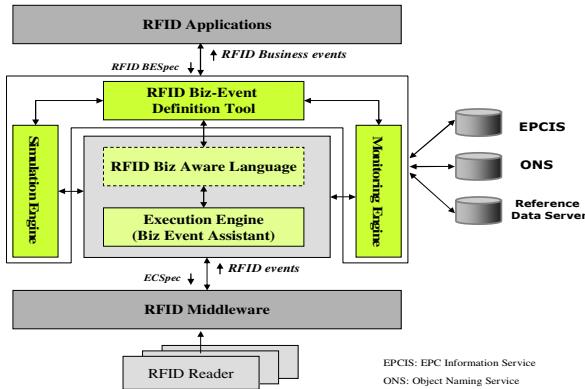
data server to retrieve reference data of RFID events, and process business logic to implement the RFID applications. Moreover, application developers must be conversant with RFID technology and communication techniques; substantial applications should involve additional codes, rather than simply business logic, to process RFID events. To maximize the benefits of RFID technology, with minimal applications impact, a separate layer that manages RFID events is required.

This research has been conducted as part of the Korean national project to develop the next generation of logistics information technology [5]. The research center has developed a prototype, version 1.0, of the Logistics Information Technology (LIT) RFID system, which was implemented on the basis of the Application Level Event (ALE) Specification [6] proposed by EPCglobal [7]. In this paper, we propose an RFID business aware framework that is located between the RFID middleware and the application. The framework combines multiple streams of RFID events with business rules and processes these to show more complex events, which have significant business meaning that can then be dispatched to the appropriate application. The framework consists of an RFID business aware language, a business aware assistant, a business event definition tool, a business event monitoring engine, and a simulation engine. The user-defined business events model specified by using a business event definition tool is converted into text type XML based business aware language and then is executed by the business event assistant. Changed RFID-related business rules are processed in this framework level, not the application level.

## 2 RFID Business Aware Framework

An *event* is defined as an object that is a record of an activity in a system [8]. An event may have particular data components. An RFID event is defined as an event caused by RFID middleware. An RFID business event is defined as that which is derived from the simple RFID event; conceptually it is a combination of an RFID event, reference data, and business rule. The RFID biz aware framework is a means of achieving transformation from RFID events to RFID business events. The transformation processes consist of a small number of activities that collect RFID events, retrieve reference data, analyze the corresponding business rule, and generate the events. Fig. 1 shows the overall architecture and the core components of the RFID business aware framework. This framework uses RFID middleware that refers to an implementation of the Application Level Event (ALE) Specification proposed by EPCglobal. In an RFID system without the RFID biz aware framework, an application sends Event Cycle Specifications (ECSpec) to the RFID middleware to request the current tags at a reader. In response to the ECSpec, the middleware returns an ECReport, typically a list of the tags currently at the reader. In an RFID system with the RFID biz aware framework, an application sends business event specifications (BCSpec) to the framework to request business events, integrating business rules into current tags at a reader. The framework parses the BESpec, composes ECSpec, and sends it to the RFID middleware. The framework receives the ECReport (referred to as the RFID event in Fig. 1) from the RFID middleware, and

processes the RFID events according to the business rules described in BC Spec. In response to the BE Spec, the RFID biz aware framework returns business events with their corresponding data.



**Fig. 1.** RFID Business aware framework architecture

## 2.1 RFID Business Aware Language

The RFID business aware language (Biz AL) is an XML-based language to describe the BE Spec [9]. This language is composed of declarative statements that specify RFID business events at a high level of abstraction without dealing with the implementation detail. That is, it can specify what has to be done but not how. In Biz AL, an *activity* is a generic unit of work that is defined to generate a business event. The *activity* is specified as either a *declaration activity* to define the data variable, a *trigger activity* to collect RFID events, a *reference activity* to retrieve reference data, or a *rule activity* to generate business events. The rule activity is comprised of a condition and a generation; it represents a business rule, which is required in the applications. The business rule constrains some aspects of the business related to the RFID event and the reference data. The generation defines processes that notify the application of the subscribed business events or specify the invocation of actions in response to an event. If conditions of the rule are not satisfied, the rule execution notifies an exception; the contents of the notification include the RFID business event name, the result of the corresponding business rule, and a related data component.

## 2.2 RFID Business Event Assistant

The RFID business event assistant (Biz EA) provides the means of processing BE Spec that are described by Biz AL. It is designed for use on the RFID middleware proposed by EPCglobal. Each activity in Biz AL is mapped to components in Biz EA. Biz EA parses the BE Spec, subscribes it to the middleware, processes multiple streams of raw RFID events, and manages the flow of activities. The progress of the

activities flow is controlled by the process variable and the transition condition. Biz EA contains sophisticated logic to process the streaming event data over the business rules established in the BESpec.

### 2.3 RFID Business Event Definition Tool

The RFID business event definition tool (Biz EDT) provides drag-and-drop support to define activities and their flows and to generate business events by a Graphical User Interface. Each activity can be expressed as a visual notation in Biz EDT. In this environment, business events are specified as sets of graphical models with textual complements. Fig. 2 shows the graphical user interface of Biz EDT after the developer has opened a BESpec file. Biz EDT consists of five panes: the left pane represents a BESpec structure as a tree, the lower left pane specifies the property of each activity, the middle upper pane models the business events, the lower middle pane shows the states of the selected activity, and the right pane presents the icons of the activities. The specification of each activity is described using the activity-property window (the popup window on the left pane, Fig. 2). When application developers describe a BESpec, this tool gives them the advantages of improved visibility, productivity, maintainability and accuracy. Biz EDT has core functions as follows:

- ***Setting Environment***

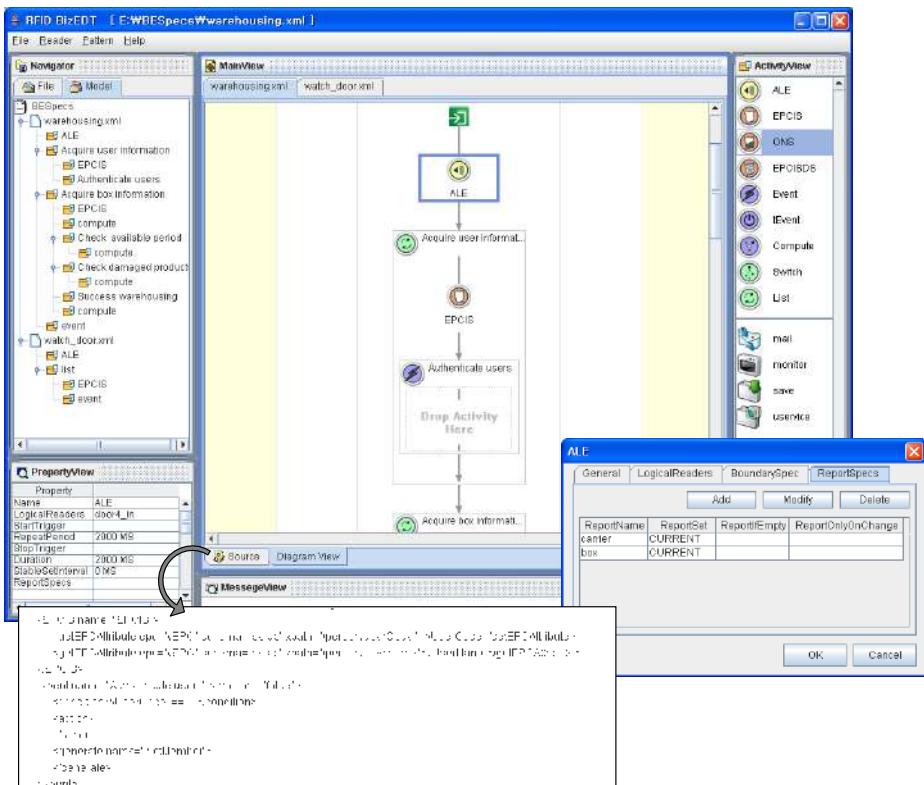
The logical names of the RFID readers are managed by the Reader Manager. After a developer enrolls an available logical reader's name when defining activities and setting properties of activities, this can be used, and the value of Variables, with their types, such as integer, string, EPC, and tag provided in Biz AL, can be set in the Variable Dialogue. The defined variables can also be selected as a developer describes the properties of each activity through the variable list.

- ***Modeling of Activities***

As shown in the center pane of Fig. 2, BESpec, which is comprised of a set of activities, is modeled with the activity icons. An icon can be dragged and dropped into an acceptable part within the model view. The model may be newly generated in the GUI view or may be converted from a BESpec source that will be shown in another tab of the center pane. The properties of each activity can be set through an activity dialogue, which clearly provides the boundary value of the property to the developer. Therefore, a developer can select allowable logical readers, variables, values, and operations to define activities.

- ***Auto-Generation of Business Event Specification***

The source window of Biz EDT (the below part in Fig. 2) shows an example of a BESpec source described by the XML-based Biz AL. This specification is generated automatically, simultaneously with the modeling BESpec. Conversely, when the BESpec is modified by a developer, the BESpec model is also changed. That is, the Biz EDT continuously updates the BESpec source according to changes in the model so that the model and BESpec source are always consistent.



**Fig. 2.** Graphical User Interface of the Business Event Definition Tool

- ***Reuse of business event model***

Design reuse is the main benefit in using frameworks. The activity icons in the BESpec model can be reused in other BESpec models. Moreover, in a similar business context, the flow and the property of the activities can be identified as patterns. Biz EDT contains business event patterns extracted from the RFID warehouse management system, u-PNU library system, RFID blood management system, etc. For example, in the warehousing business context of the warehouse management system, there may be business event patterns, such as checking for non conforming products, checking the available period of products, checking the product information with the purchase order specification, etc.

## 2.4 Business Event Monitoring Engine

The business event monitoring engine connects with an arbitrary business event assistant in operation and displays a visual representation of the event data. The monitoring engine provides information about the streaming RFID events from the middleware, the exception conditions specified in the business rules, and the business actions corresponding to the business rules. The monitoring engine has two types of monitoring, console and file type. A business event can be checked in a monitoring

window in real time and in the file. A developer can verify the business events from BESpec in advance of applying to the RFID applications. The engine consults the business rules to evaluate what corrective actions are best suited for checking automatically.

## 2.5 Business Event Simulation Engine

The RFID system can be tested in the execution environment with physical hardware (RFID reader and tags) on the network. The business event simulation engine simulates the RFID readers that generate data and events and provides an effective means of evaluating alternative business rules in the BESpec. In addition, the simulation engine contains an abstraction of the actual external system that forms the simulation, including EPCIS servers as a suite of Web services companies are expected to use in managing their EPC data, databases, etc. The simulation engine, together with the capability of the monitoring engine, improves the efficiency of business actions and decisions by responding to business rule conditions specified in the BESpec. Ultimately, it makes the deployment of the RFID system quick and cost effective.

## 3 Related Works

Vendors, such as Sun Microsystems [1], IBM [2], Oracle [3], and Microsoft [4], have been extending their application development and middleware technology stacks to handle RFID. These middleware systems delete duplicate readings of the same tag and help manage the flow of data. Several research groups are attempting to derive meaningful context information from raw data acquired by sensors. Recent researches have focused on providing infrastructure support for context-aware systems. Ranganathan and Campbell proposed middleware that facilitates the development of context-aware agents [10]. Reconfigurable Context-Sensitive Middleware facilitates the development and runtime operations of context-sensitive pervasive computing software [11]. Gu developed a service-oriented middleware that provides support to acquire, discover, interpret and accesses various contexts to build context-aware services [12]. These middleware are for general sensors, and therefore do not address various characteristics of the RFID technology. Information representing business rules has traditionally been embedded in application codes and database structures. To the best our knowledge, few existing approaches focus on an RFID technology integration of business rules. RuleBAM [13] is an architectural framework that supports the definition of business activity management (BAM) policy, generates business rules instances, and integrates business rules into the target system.

## 4 Conclusions and Future Work

To realize the impact of RFID on business processes, raw RFID events can be translated into one or more business events and configure these for appropriate business applications. In addition, business rules should be modeled and managed as separate entities to reduce the impact of changes in the business logic. This paper

described an RFID business aware framework that enables a user to specify and update the business rules without changing the source code of the application. The framework processes a chain of activities that control processing the RFID business event request. Using this framework, we have developed several RFID-enabled applications. As a result, RFID-enabled applications do not have to involve additional code to process RFID events, thereby substantially reducing the cost of developing and managing RFID applications. Currently, we are in the process of working with a company on implementing an RFID-enabled logistics based on middleware and the proposed framework. Our future research activities include extension of the RFID business aware framework, which is able to process RFID readers and other types of sensors, including temperature, humidity, shock, location, etc.

## References

- [1] Sun Microsystems, <http://www.sun.com/software/solutions/rfid/>
- [2] IBM, [http://www306.ibm.com/software/pervasive/w\\_rfid\\_premises\\_server/](http://www306.ibm.com/software/pervasive/w_rfid_premises_server/), December 2004.
- [3] Oracle, [http://www.oracle.com/technology/products/iaswe/edge\\_server](http://www.oracle.com/technology/products/iaswe/edge_server)
- [4] Microsoft, <http://www.microsoft.com/business/insights/about/aboutus.aspx>
- [5] Research Center for Logistics Information Technology, <http://www.rclit.com/>
- [6] EPCglobal, The Application Level Events (ALE) Specification Version 1.0, September 2005.
- [7] EPCglobal, <http://epcglobalus.gs1us.org/>
- [8] Luckham, D., *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, Addison-Wesley, ISBN 0-201-72789-7, 2002.
- [9] M. Moon, Y. Kim, and K. Yeom, “Contextual Events Framework in RFID System”, In proceedings of third International Conference on Information Technology (IEEE Computer Society) pp. 586-587, 2006.
- [10] A. Ranganathan and R.H. Campbell, “A Middleware for Context-Aware Agents in Ubiquitous Computing Environments”, In proceedings of International Middleware conference, LNCS Vol. 2672, pp.143-161, 2003.
- [11] S. Yau, F. Karim, Y. Wang, B. Wang, and S. Gupta, “Reconfigurable Context-Sensitive Middleware for Pervasive Computing”, IEEE Pervasive Computing, Vol. 1, No. 3, pp. 33-40, 2002.
- [12] T. Gu, H.K. Pung, and D.Q. ZJang, “A Service-oriented middleware for building context-aware services”, Journal of Network and Computer Applications (JNCA), Vol.28, No. 1, pp.1-18, 2005.
- [13] J. Jeng, D. Flaxer, and S. Kapoor, “RuleBAM: A Rule-Based Framework for Business Activity Management”, In proceedings of the 2004 IEEE International Conference on Services Computing (SCC’04), pp. 262-270, 2004.

# Approximate Similarity Search over Multiple Stream Time Series

Xiang Lian<sup>1</sup>, Lei Chen<sup>1</sup>, and Bin Wang<sup>2</sup>

<sup>1</sup> Hong Kong University of Science and Technology Kowloon, Hong Kong, China

{xlian, leichen}@cse.ust.hk

<sup>2</sup> Northeastern University, China

binwang@mail.neu.edu.cn

**Abstract.** Similarity search over stream time series has a wide spectrum of applications. Most previous work in static time-series databases and stream time series aim at retrieving the *exact* answer to a similarity search. However, little work considers the *approximate* similarity search in stream time series. In this paper, we propose a *weighted locality-sensitive hashing* (WLSH) technique, which is adaptive to characteristics of stream data, to answer *approximate* similarity search over stream time series. Due to the unique requirement of stream processing, we present an efficient method to update hash functions adaptive to stream data and maintain hash files incrementally at a low cost. Extensive experiments demonstrate the effectiveness of WLSH, as well as the efficiency of *approximate* similarity search via hashing on stream time series.

**Keywords:** approximate similarity search, weighted locality-sensitive hashing.

## 1 Introduction

Similarity search over stream time series has many applications such as Internet traffic analysis [2], sensor network monitoring [7], moving object search [1], and financial data analysis [6]. In particular, a typical *similarity query* retrieves subsequences from stream time series that are similar to a user-specified query time series. In general, similarity search can be classified into two categories, *exact* and *approximate* similarity searches. The former category obtains the *exact* answer to queries without *false dismissals*, whereas the latter retrieves *approximate* ones by allowing some *false negatives* with a certain precision.

Existing work on *approximate* similarity search in static time-series databases include *approximate nearest neighbor* search via *locality-sensitive hashing* (LSH) [3], which retrieves *nearest neighbors* of a query time series approximately. To the best of our knowledge, however, there does not exist any previous work in *stream time series* on such problem. In this paper, we propose a novel hashing approach, WLSH, to answer the *approximate range queries* over stream time series. We make the following contributions:

1. We propose in Section 3 a general framework for *approximate* similarity search via hashing over multiple stream time series.

2. We generalize in Section 4 the *locality-sensitive hashing* (LSH) approach in static time-series databases [3] to the *weighted locality-sensitive hashing* (WLSH), adaptive to data characteristics.
3. We illustrate in Section 4 the incremental update of hash functions, which are adaptive to stream data, and the maintenance of hashing files.

Section 2 reviews the *approximate* similarity search over traditional time-series databases. Section 5 demonstrates the performance of our proposed approach through extensive experiments. Finally, Section 6 concludes this paper.

## 2 Related Work

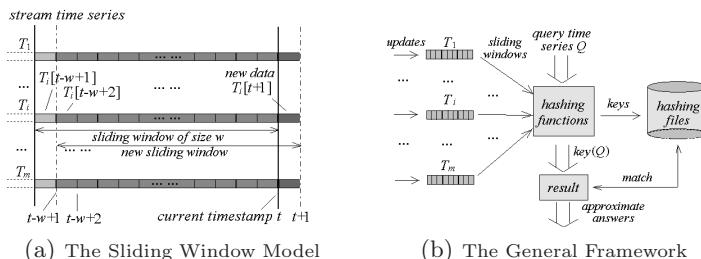
In this section, we briefly overview the *approximate* similarity search in static time-series databases. In particular, Gionis et al. [3] present a *locality sensitive hashing* (LSH) approach to answer similarity queries *approximately*. As a consequence, the retrieved series are similar to a query with a guaranteed probability. Specifically, two assumptions of similarity search over time-series databases are made, that is,  $L_1$ -norm distance is used to measure the similarity between two time series and all values in any time series are positive integers. The LSH method [4] first converts each subsequence  $T$  of length  $w$  into a bit vector  $V(T)$  containing  $w'$  bits in the *Hamming* space, where  $w' = (C \cdot w)$  and  $C$  is the ceiling in the domain of values in  $T$ . In particular, the vector  $V(T)$  is obtained by concatenating the bit representation  $V(T[i])$  of each value  $T[i]$  ( $0 \leq i \leq w - 1$ ), where  $V(T[i])$  is a sequence of  $T[i]$  “1” bits followed by  $(C - T[i])$  “0” bits. Then, an LSH function  $H_j \in \mathcal{H}$  is obtained by randomly selecting  $k$  positions in a bit vector with  $w'$  bits. Without loss of generality, assume  $j_1, j_2, \dots$ , and  $j_k$  are  $k$  positions randomly selected by  $H_j$ , whose inputs are bit vectors  $V(T)$  of  $T$  with  $w'$  positions and outputs  $key_j(T)$  the concatenations of  $k$  bits from  $k$  positions in  $V(T)$ , respectively. Given two bit vectors  $V(T)$  and  $V(T')$  with the same length  $w'$ , converted from time series  $T$  and  $T'$ , respectively, if  $dist(T, T') \leq \varepsilon$ , then it holds that  $key_j(T) = key_j(T')$  with a probability greater than  $(1 - \frac{\varepsilon}{w'})$ , where  $dist$  is an  $L_1$ -norm distance function. In other words, if two time series are similar, then it is very likely that they have the same key. This is because the  $L_1$ -norm distance between two series is exactly the number of bit differences in their bit vectors. Two similar bit vectors have fewer bit differences and thus share the same key with higher probability. Given any query time series  $Q$ , we first obtain its keys using the same set of hash functions and then retrieve as candidates all the content in buckets where keys are located. Finally, candidates are refined by checking their real distances to the query. To the best of our knowledge, however, there is no previous work on *approximate* similarity search in *stream time series*. Motivated by this, we introduce the problem of *approximate* similarity search among multiple stream time series and a general framework for using the hashing method to answer similarity queries.

### 3 Problem Definition

Fig. 1(a) illustrates the sliding window model with  $m$  stream time series  $T_1, T_2, \dots$ , and  $T_m$ . For each series  $T_i$ , we consider the most recent  $w$  data  $T_i[t-w+1], T_i[t-w+2], \dots$ , and  $T_i[t]$  within a *sliding window*  $T_i[t-w+1 : t]$  at the current timestamp  $t$ . Note that, we use  $L_1$ -norm distance to measure the similarity between two series, and assume that  $T_i[j]$  is a positive integer for any  $j$ , following the same assumptions in [3]. Given any query time series  $Q$  of length  $w$ , a *similarity query* retrieves those subsequences (*sliding windows*)  $T_i[t-w+1 : t]$  from stream time series  $T_i$  such that  $\text{dist}(T_i[t-w+1 : t], Q) \leq \varepsilon$ , where  $1 \leq i \leq m$  and  $\text{dist}$  is an  $L_1$ -norm distance between two series. Next, at timestamp  $(t+1)$ , each series  $T_i$  receives a new data element  $T_i[t+1]$ , while the old data  $T_i[t-w+1]$  is out of date and discarded. Therefore, similarity queries at timestamp  $(t+1)$  are performed on new *sliding windows*  $T_i[t-w+2 : t+1]$  for all  $i$   $1 \leq i \leq m$ .

Fig. 1(b) illustrates the general framework of our hashing method to answer *approximate* similarity search over the sliding window model (Fig. 1(a)) of multiple stream time series. Specifically, weighted *locality-sensitive hashing* (LSH) functions are applied to hash subsequences from stream time series into keys. In particular, with each hash function  $H_j \in \mathcal{H} (1 \leq j \leq l)$ , we hash  $m$  most recent sliding windows  $T_i[t-w+1 : t]$  of size  $w$  from  $m$  stream time series  $T_i$ , respectively, into  $m$  keys, which are then inserted into a hash file  $HF_j$  together with their stream id's. Given any query series  $Q$ , we first obtain the hash key  $\text{key}(Q)$  of  $Q$  using a hash function  $H_j \in \mathcal{H}$  and retrieve all the stream id's as candidates in the bucket of  $HF_j$  where  $\text{key}(Q)$  is located. Finally, each resulting candidate is further refined. Since our hash functions are locality-sensitive and adaptive to stream data, the final answer can achieve high query accuracy.

We focus on two issues with respect to query processing and file organization in our framework. First, we want to *dynamically* choose hash functions to achieve high query accuracy that are adaptive to stream data. Moreover, we want the resulting data in buckets of hash files to be of approximately equal size, in order to achieve low query processing cost. Second, hash files is desired to have low update cost. Typically, we consider the *incremental* maintenance of hash files.



**Fig. 1.** Illustration of Our Problem

## 4 Approximate Similarity Search Via Hashing

### 4.1 Weighted Locality-Sensitive Hashing (WLSH)

In this section, we propose a variant of the *locality-sensitive hashing* (LSH) technique, *weighted locality-sensitive hashing* (WLSH), which adapts to the underlying data characteristics and achieves both high query accuracy and efficiency. Recall that, the LSH method [4] always selects  $k$  random positions in the bit vector *uniformly*, which is independent on the underlying data characteristics. Instead, we propose a *weighted locality-sensitive hashing* (WLSH) approach, which selects  $k$  positions in bit vectors with *weighted* probabilities. Consider one specific position  $s$  in bit vectors of a data set, which is chosen as part of the key in a hash function  $H_j$ . We have the lemma as follows:

**Lemma 1.** *Let  $V(T)$  and  $V(T')$  be two bit vectors randomly selected from a data set. The probability that bits  $V(T)[s]$  and  $V(T')[s]$  are the same is  $\left(1 - \frac{2n_0n_1}{(n_0+n_1)^2}\right)$ , where  $s$  is a selected position in the hash function  $H_j$ ,  $n_0$  the number of “0” bits and  $n_1$  that of “1” bits, for the  $s$ -th position of all vectors in the data set.*

Note that,  $\frac{2n_0n_1}{(n_0+n_1)^2}$  in Lemma 1 is exactly two times the bit variance  $\sigma_s^2$  at the  $s$ -th position for all vectors. Therefore, if we increase the chance of selecting a position with small bit variance in hash function  $H_j$ , the probability that two similar series have the same key value is higher than LSH. That is, we can improve the *locality* of any two similar series (query accuracy) by giving *high* probability (weight) to positions with *small* bit variance. On the other hand, however, we want to have low query processing cost. Thus, choosing those positions with high bit variances can result in *uniform* bucket size, whose searching cost is low. Based on these two observations, our WLSH method makes a trade-off between high query accuracy and low query processing cost. Specifically, WLSH selects the  $s$ -th position in the bit vector with the probability  $f_s$  proportional to  $p \cdot \sigma_{\min}^2 / (\sigma_s^2) + (1 - p) \cdot \sigma_s^2 / (\sigma_{\max}^2)$ , where  $p \in [0, 1]$  is a trade-off parameter, and  $\sigma_s^2$  is the bit variance at the  $s$ -th position,  $\sigma_{\min}^2$  and  $\sigma_{\max}^2$  the minimum and maximum possible bit variances, respectively.

### 4.2 Dynamic Maintenance of Hash Functions in Stream Time Series

Next, we apply WLSH to the stream time series scenario. Specifically, in the sliding window model (Fig. 1(a)), we always maintain a hash file containing the most recent sliding window  $T_i[t-w+1:t]$  from each stream time series  $T_i$ , whose key  $key_j(T_i[t-w+1:t])$  is obtained by using a WLSH function  $H_j(t)$  at timestamp  $t$ . For brevity, we denote the hash function  $H_j(t)$  by  $H(t)$  and the key  $key_j(T_i[t-w+1:t])$  by  $key(T_i, t)$ . Then, at the next timestamp  $(t+1)$ , each stream series  $T_i$  receives a new data  $T_i[t+1]$  and discards the out-of-date one  $T_i[t-w+1]$ . Since more positions from new bits may be included and the expired ones removed in hash function  $H(t)$ , we have to incrementally update the hash function from  $H(t)$  to  $H(t+1)$ , illustrated as follows.

**Initialization.** Up to timestamp  $(w - 1)$ , each stream time series  $T_i$  has received a window  $T_i[0 : w - 1]$  of  $w$  data elements. Initially, we compute the bit variance of each position in bit vectors of all subsequences (windows), and obtain the weight  $f_s$  of each position  $s$ , where  $0 \leq s \leq w' - 1$ , as mentioned in Section 4.1. Next, the hash function  $H(w - 1)$  randomly selects  $k$  out of  $w'$  positions such that each position  $s$  is selected with the probability  $f_s / (\sum_{j=0}^{w'-1} f_j)$ .

**Incremental Update.** Using the same example as in the initialization step, at the next timestamp  $w$ , due to the insertion and deletion of data,  $C$  new positions ( $V(T_i)[w' : w' + C]$ ) arrive and  $C$  old ones ( $V(T_i)[0 : C - 1]$ ) are discarded. Similarly, we can obtain weights  $f_{w'}, f_{w'+1}, \dots, f_{w'+C-1}$  of  $C$  new positions. One straightforward way to obtain the new hash function is to re-select  $k$  positions from scratch. However, this results in totally different bits in hash keys from the previous timestamp and thus the hash file needs to be re-organized at high cost. Motivated by this, we want to utilize as many positions as possible that have already been selected in the window  $V(T_i[0 : w - 1])$ . We consider three cases. For Case 1,  $\sum_{j=0}^{C-1} f_j = \sum_{j=w'}^{w'+C-1} f_j$ . Since weights of new and old data are the same, the same number of bits is selected in the new data as that in old ones. For Case 2,  $\sum_{j=0}^{C-1} f_j < \sum_{j=w'}^{w'+C-1} f_j$ . We deselect those positions from  $C$  to  $w' - 1$  with probability  $(1 - (\sum_{j=0}^{w'-1} f_j)) / (\sum_{j=C}^{w'+C-1} f_j)$ . For the new data from position  $w'$  to  $(w' + C - 1)$ , we select the  $s$ -th position with probability  $f_s / (\sum_{j=C}^{w'+C-1} f_j)$ . For Case 3,  $\sum_{j=0}^{C-1} f_j > \sum_{j=w'}^{w'+C-1} f_j$ . We first select  $size_{old} \cdot (\sum_{j=w'}^{w'+C-1} f_j) / (\sum_{j=0}^{C-1} f_j)$  positions for the new data. The remaining bits for positions from  $C$  to  $w' - 1$  are selected with probability for the  $s$ -th position  $f_s / (\sum_{j=C}^{w'+C-1} f_j)$ .

Finally, since each update within the *sliding window* may result in a new hash function, say from  $H(t)$  to  $H(t + 1)$ , hash keys of updated *sliding windows* (subsequences) may also change over time. However, due to the re-use of those already selected  $k$  positions in the hash function, only partial bits in keys have changed (Section 4.2), which significantly reduces the hash file maintenance cost.

## 5 Experimental Evaluation

We evaluate the performance of our hashing approach WLSH on both real and synthetic data sets, *sstock* and *randomwalk*, respectively, whose values are normalized to the interval  $[1, 100]$  (similar to [3]). The  $L_1$ -norm is used as the distance function.

### 5.1 Performance of LSH vs. WLSH

First, we compare the effectiveness of LSH with WLSH to answer *approximate range queries* on static data sets, in terms of the *recall ratio*, which is the number of the actual answer in the candidate set divided by the total number of candidates. We randomly extract 100 series of length 100 from data sets as queries.

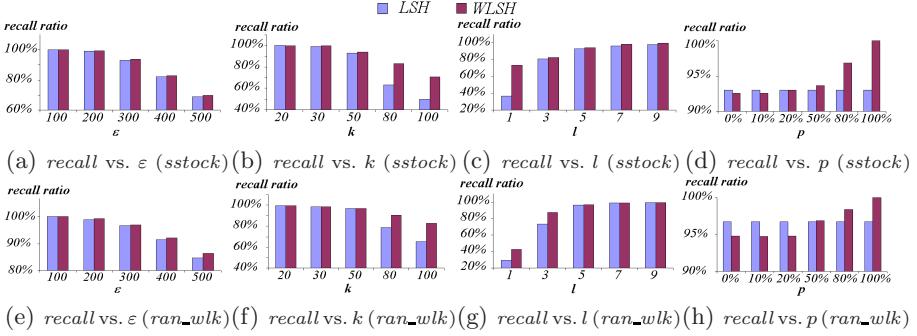
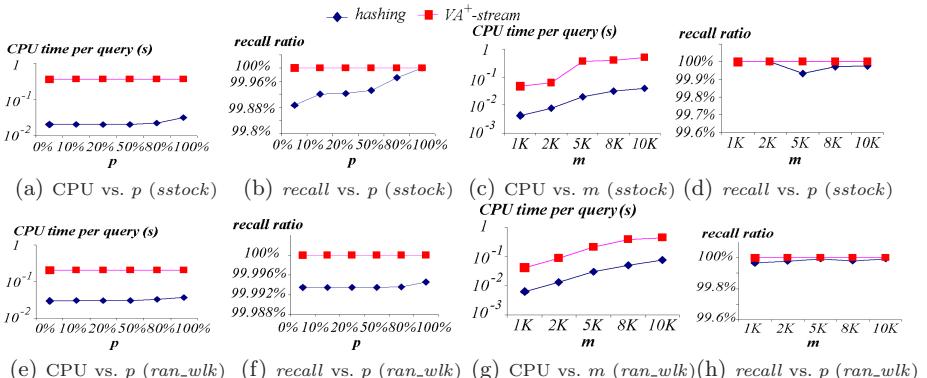
**Fig. 2.** Performance of WLSH vs. LSH

Fig. 2 illustrates the comparison of *recall ratios* of LSH vs. WLSH under different parameters  $\varepsilon$ ,  $k$ ,  $l$ , and  $p$ , using both real *sstock* and synthetic *randomwalk* data sets. Specifically, each time we vary one parameter by assigning *default values* to other parameters (i.e.  $\varepsilon = 300$ ,  $m = 5K$ ,  $k = 50$ ,  $l=5$ , and  $p = 50\%$ ). Changing parameters  $\varepsilon$ ,  $k$  and  $l$ , WLSH always performs better than LSH having higher *recall ratios* where  $p = 50\%$ . With different trade-off  $p$  values, however, small  $p$  gives a lower *recall ratio* than LSH, and nevertheless lower cost of query processing than LSH, presented later.

## 5.2 Performance of Our Hashing Approach vs. VA<sup>+</sup>-Stream

Next, Fig. 3 illustrates both the query efficiency and accuracy of our proposed hashing approach, compared to *exact* similarity search method, VA<sup>+</sup>-stream [5], over multiple stream time series on *sstock* and *randomwalk*. Specifically, we consider sliding windows of size  $w$  ( $= 100$ ) in streams. In general, our hashing approach outperforms VA<sup>+</sup>-stream requiring an order of magnitude less CPU

**Fig. 3.** Performance of Our Hashing Approach vs. VA<sup>+</sup>-Stream

time per query, and yet achieving high *recall ratio* very close to 100% (in VA<sup>+</sup>-stream). Since update costs of incremental hash file in our approach and VA<sup>+</sup>-stream are similar (i.e.  $O(m)$  cost), results are omitted due to the space limit.

## 6 Conclusions

Similarity search over dynamic stream time series has a wide spectrum of applications. Previous work studies the *approximate* similarity search on static databases. In this paper, we consider the same problem in the scenario of stream time series. Specifically, we propose a novel WLSH approach to map each series to a key, with which similarity queries can be answered with high query accuracy and low update and query processing cost. Extensive experiments have verified the performance of our method, compared to the *exact* one, VA<sup>+</sup>-stream.

## References

1. L. Chen et al. Robust and fast similarity search for moving object trajectories. *SIGMOD*, 2005.
2. C. Cranor et al. Gigascope: a stream database for network applications. *SIGMOD*, 2003.
3. A. Gionis et al. Similarity search in high dimensions via hashing. *VLDB*, 1999.
4. P. Indyk, R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *STOC*, 1998.
5. X. Liu, H. Ferhatosmanoglu. Efficient  $k$ NN search on streaming data series. *SSTD*, 2003.
6. H. Wu et al. Online event-driven subsequence matching over financial data streams. *SIGMOD*, 2004.
7. Y. Zhu, D. Shasha. Efficient elastic burst detection in data streams. *SIGKDD*, 2003.

# WT-Heuristics: A Heuristic Method for Efficient Operator Ordering

Jun-Ki Min

School of Internet-Media Engineering,  
Korea University of Technology and Education,  
Byeongcheon-myeon, Cheonan, Chungnam, Republic of Korea, 330-708  
`jkmin@kut.ac.kr`

**Abstract.** In this paper, we focus on the processing of stream data whose characteristics vary unpredictably over time. Particularly, we suggest a method which generates an efficient operator execution order called WT-Heuristic. Our method changes the execution order with respect to the change of data characteristics with minimum overheads.

## 1 Introduction

This paper deals with the operator ordering problem for stream data whose properties vary over time. For brevity, we assume that a query consists of a set of commutative filters (i.e., operators) like most related literature [12]. When a tuple  $t$  is inserted into a filter, a filter drops  $t$  or outputs  $t$  with respect to the predefined condition for a filter. Overall processing costs can vary widely across different filter ordering. For example, operator  $O_1$  drops tuples whose values are 1, 3, and 5 in a unit time, as well as operator  $O_2$  drops tuples whose values are 2, 4, and 6 in a unit time. Let an input stream be 2, 4, 6. If an operator order is  $O_1$  and  $O_2$ , then the overall cost is 6 unit times. Otherwise, the cost is 3 unit times.

Commutative filters are very common in stream application [34]. In the work of [2], the authors show that the operator order problem is applicable to the ordering problem for wide class of multiway joins.

## 2 Related Work

In the work of [2], the A-Greedy technique for operator ordering was proposed. In the A-Greedy technique, when the operator ordering is  $O_{f(1)}, O_{f(2)}, \dots, O_{f(n)}$ , the query cost  $C$  can be formalized as the follow:

$$\sum_{i=1}^n (t_i \cdot D_i), \text{ where } D_i = \begin{cases} 1 & (i = 1) \\ \prod_{j=1}^{i-1} (1 - d(j|j-1)) & (i > 1) \end{cases} \quad (1)$$

In Equation (1),  $d(i|j)$  denotes the conditional probability that  $O_{f(i)}$  will drop a tuple  $e$  from input stream  $I$ , given that  $e$  was not dropped by any of  $O_{f(1)}$ ,

$O_{f(2)}, \dots, O_{f(j)}$ . And,  $t_i$  represents the expect time for  $O_{f(i)}$  to process one tuple. Thus,  $C$  is the average time to process (or drop) an incoming tuple. The goal of the A-Greedy technique is to find efficient orderings that minimize  $C$ . In order to achieve their goal, the A-Greedy technique uses a greedy heuristic rule which rearranges the operator order satisfying the following formula:

$$\frac{d(i|i-1)}{t_i} \geq \frac{d(j|i-1)}{t_j}, 1 \leq i < j \leq n \quad (2)$$

In order to apply the greedy heuristics, A-Greedy uses a profiling technique. The profile is a sliding window of profile tuples. In profiling, a tuple  $e$  which was dropped during processing is selected with probability  $p$ , called the *drop-profiling probability*. Then, the A-Greedy profiler artificially applies  $e$  to all operators and generates a profiler tuple whose attribute  $b_i$  is 1 if  $O_i$  drops  $e$  and  $b_i = 0$  otherwise (see Figure 1-(a)).

$b_1 \ b_2 \ b_3 \ b_4$				$O_4 \ O_1 \ O_3 \ O_2$			
1	0	0	0	4	2	3	1
0	0	1	1		1	0	0
0	1	1	1			0	0
1	0	1	1				0
0	0	0	1				

(a) Profile window

(b) Matrix View

**Fig. 1.** A-Greedy profile

The A-Greedy reoptimizer, which keeps the operator order to obey the greedy heuristic rule, maintains a specific view (see Figure 1-(b)) using the profile window. As shown in the first row of Figure 1-(b),  $O_4$  drops the most tuples. Thus, if the processing costs of all operators are equal,  $O_4$  is the first operator. The second row reports the numbers of tuples which are not dropped by  $O_4$  but dropped by  $O_1$ ,  $O_3$ , and  $O_2$ , respectively. Using this manner, A-Greedy arranges the operator order. When the profile changes, the reoptimizer computes a new operator order.

The problem of the A-Greedy technique is that the profiling overhead is large. A normal tuple may be dropped by an operator, but a tuple for profiling is applied to all operators. The cost of a normal tuple is Equation-(1). But the cost of a tuple for profile is  $\sum_{i=1}^n t_i$  which is grater than or equal to Equation-(1).

### 3 WT-Heuristics

#### 3.1 Goal of WT-Heuristics

The purpose of query processing is to process input data and generate output efficiently. Thus, the time for processing a tuple which will not be in results should be minimized.

Let the selectivity of the first operator  $O_1$  be  $s_1$  and the processing cost of  $O_1$  be  $t_1$ . If a tuple is sent to  $O_1$ , the expected cost for dropping a tuple is  $(1 - s_1) \cdot t_1$ . And, let the processing cost of the second operator  $O_2$  be  $t_2$  and the conditional selectivity of  $O_2$  be  $s_2$  which denotes the probability that  $O_2$  will not drop a tuple  $e$ , given that  $e$  was not dropped by  $O_1$ . Then, the waste time to drop a tuple  $e$  on  $O_2$  is  $(1 - s_2) \cdot (t_1 + t_2)$ . Thus, the total waste time  $W$  can be defined as the follow:

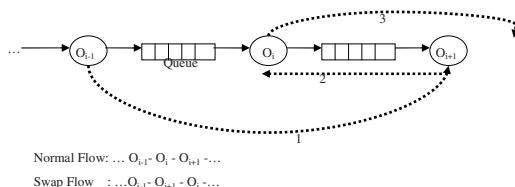
$$W = \sum_{i=1}^n ((1 - s(i|0, 1, \dots, i-1)) \cdot \sum_{j=1}^i t_j) \quad (3)$$

In Equation-(3),  $s(i|0, 1, \dots, i-1)$  is the conditional probability that a tuple  $e$  is selected by  $O_{f(i)}$ , given  $e$  is not dropped by any  $O_{f(0)}, O_{f(1)}, O_{f(2)}, \dots, O_{f(i-1)}$ .  $t_j$  is the processing cost of an operator  $O_{f(j)}$  to process a tuple.

### 3.2 WT-Heuristics

Assume that a query consists of  $n$  commutative operators (i.e., filters) and the current operator order is  $O_0, O_1, \dots, O_n$ , where  $O_0$  denotes the stream source itself. Each operator  $O_i$  ( $1 \leq i \leq n$ ) keeps its processing cost  $t_i$  and conditional selectivity  $s(i|0, 1, \dots, i-1)$ . In addition, the conditional probability  $s(i+1|0, 1, \dots, i-1)$  is estimated. To obtain  $s(i+1|0, 1, \dots, i-1)$  in WT-heuristics, when a tuple  $e$  is not dropped by an operator  $O_{i-1}$  ( $1 \leq i \leq n-1$ ),  $e$  is chosen with probability  $p$ , called *swap-probability*. If  $e$  is chosen,  $e$  is transmitted to operator  $O_{i+1}$ , instead  $O_i$ . By using this, we can estimate  $s(i+1|0, 1, \dots, i-1)$ . Also, if the chosen tuple  $e$  is not dropped by  $O_{i+1}$ , the tuple  $e$  is sent to the operator  $O_i$ . Thus, we can estimate a conditional probability  $s(i|0, 1, \dots, i-1, i+1)$ .

As shown in Figure 2, general tuples are processed following the current operator order,  $\dots, O_{i-1}, O_i, O_{i+1}, \dots$ . In contrast, a tuple  $e$  with swap-probability  $p$  is processed following the order,  $\dots, O_{i-1}, O_{i+1}, O_i, \dots$ .



**Fig. 2.** Operator swapping

In WT-heuristics, using these conditional probabilities, the waste time  $W_{i,i+1}$  when a tuple is processed with  $O_i, O_{i+1}$ , and the waste time  $W_{i+1,i}$  with  $O_{i+1}, O_i$  can be estimated. The formulae for  $W_{i,i+1}$  and  $W_{i+1,i}$  are defined as follows:

$$W_{i,i+1} = (1 - s(i|0, 1, \dots, i-1)) \left( \sum_{j=1}^i t_j \right) + (1 - s(i+1|0, 1, \dots, i)) \left( \sum_{j=1}^{i+1} t_j \right) \quad (4)$$

$$W_{i+1,i} = (1 - s(i+1|0, 1, \dots, i-1)) \left( \sum_{j=1}^{i-1} t_j + t_{i+1} \right) + (1 - s(i|0, 1, \dots, i-1, i+1)) \left( \sum_{j=1}^{i+1} t_j \right) \quad (5)$$

Since  $W_{i,i+1}$  denotes the partial waste time on current operator order and  $W_{i+1,i}$  denotes the estimated waste time when the operators  $O_i$ , and  $O_{i+1}$  are interchanged, if  $W_{i,i+1}$  is greater than  $W_{i+1,i}$ , WT-heuristics rearranges the operator order such that  $\dots, O_{i+1}, O_i, \dots$

WT-heuristics detects situation where a swap between adjacent operators in the current operator will improve performance. Thus, the WT-heuristics technique may take much time to converge the best plan generated by the A-Greedy technique. Also, WT-heuristics may stay on local optimal plan. However, WT-heuristics has very low run-time overhead compared to A-Greedy.

If WT-heuristics reacts sensitively with respect to change of input stream's characteristics, the operator ordering can trash. Particularly, if  $W_{i,i+1}$  and  $W_{i+1,i}$  are almost equal, the operator trashing incurs. In order to avoid trashing, we borrow *trash-avoidance parameter*  $\alpha$  ( $0 < \alpha \leq 1$ ) from [2]. Thus, if  $W_{i+1,i} < \alpha \cdot W_{i,i+1}$ , two adjacent operators are interchanged.

## 4 Experiments

In this paper, we analyze the performance of our proposed method, WT-heuristics using simulation. We empirically compared the performance of WT-heuristics with A-Greedy and a static operator ordering using synthetic data sets. In our experiments, we found that WT-heuristics shows significantly better performance.

### 4.1 Experimental Environments

The experiments were performed on Pentium IV-1.7GHz platformed with Windows XP and 1GBytes of main memory.

In order to show the efficiency of WT-heuristics, we implemented three operator ordering algorithms: WT-heuristics, A-Greedy, and the static method. The static method does not change the initial operator order and the other methods changes the orders adaptively.

Synthetic data sets are generated with uniform distribution whose domain is  $[0, 10000]$ . The other characteristics of parameters used in the experiments are summarized in Table 1.

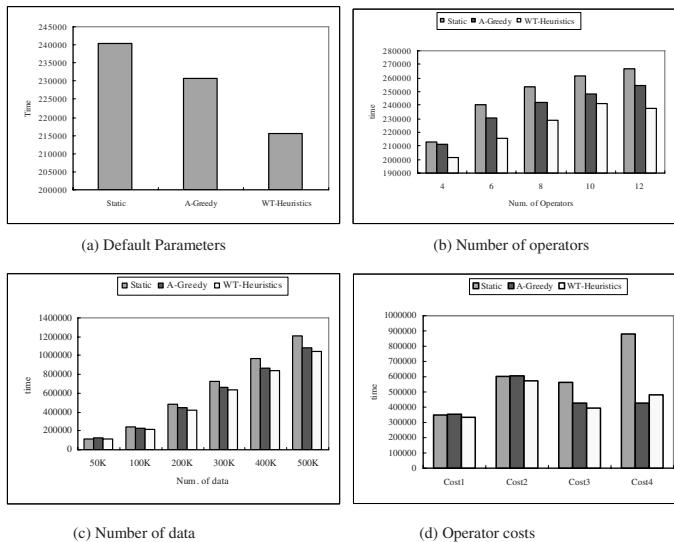
**Table 1.** Parameters

Parameter	Default Value
Number of operators	6
Number of Input Data	100000
Operator Processing Cost	1
Unconditional selectivity	50%
Correlation factor ( $\Gamma$ )	$\Gamma = 2$
Trash avoidance parameter ( $\alpha$ )	$\alpha = 0.9$
Drop-profiling probability (for A-Greedy)	0.01
Profile window Size (for A-Greedy)	500
Swap probability (for WT-heuristics)	0.01

One of factors affecting performance is the correlation among operators. To capture the correlation among operator, we used the correlation factor  $\Gamma$  introduced in [2]. The  $n$  operators are divided into  $\lceil n/\Gamma \rceil$  groups containing  $\Gamma$  operators each. Two filters are independent (but is not disjoint) if they belong to different group. If they belong to same group, they have positive correlation such that they generate the same results on 80% of input data.

## 4.2 Experimental Results

The experimental result with default parameters is presented in Figure 3-(a). Since WT-heuristics does not incur extra overheads to obtain a profile, WT-heuristics is superior to the other methods.

**Fig. 3.** Experimental results

As shown in Figure 3-(b), as the number of operators increases, the processing costs of each technique increases. But the performance ratio of each technique does not change significantly. Figure 3-(c) illustrates the processing cost

of each technique varying the number of stream data: 50,000, 100,000, 200,000, 300,000, 400,000, and 500,000. When the number of data is extremely small (i.e., 50,000), A-Greedy shows the worst performance since the profiling overhead appears but the benefit of the rearranged operator order is not shown in A-Greedy. Finally, we measure the performance of each technique varying the cost of individual operators. As Cost1, we assign the operator costs  $<1,1,1,1,5,5>$  to six operators sequentially. As Cost2, operator costs  $<1,1,5,5,5,5>$  are used as well as  $<1,5,1,5,1,5>$  for Cost3, and  $<5,1,5,1,5,1>$  for Cost4.

The performance of each technique is plotted in Figure 3(d). In Cost1, Cost2, and Cost3, WT-heuristics shows the best performance. In Cost4, WT-heuristics is worse than A-Greedy. In this experiment, we found that the final operator orders of A-Greedy and WT-heuristics are equal. As mentioned earlier, since WT-heuristics swaps adjacent operators at once, WT-heuristics may take much time to converge the best plan. The experiment of Cost4 shows this case. However, as shown in the other experimental results, since WT-heuristics does not incur the extra overhead, the performance gap is amortized over time.

Consequently, WT-heuristics shows the best performance over most of all cases.

## 5 Conclusion

In this paper, we propose WT-heuristics which modifies the operator orders in environments where changes unpredictably. The goal of WT-heuristics is to reduce the waste time. We implement our WT-heuristics and conducted extensive experimental study with synthetic data over diverse environments. Experimental results show that WT-heuristics improves the processing costs since WT-heuristics obtains the required information without the system overheads.

**Acknowledgement.** This research was supported by the Ministry of Information and Communication, Korea, under the College Information Technology Research Center Support Program, grant number IITA-2006-C1090-0603-0031.

## References

1. Avnur, R., Hellerstein, J.M.: Eddies: Continuously adaptive query processing. In: Proceedings of ACM SIGMOD Conference. (2000) 261–272
2. Babu, S., Motwani, R., Munagala, K., Nishizawa, I., Widom, J.: Adaptive ordering of pipelined stream filters. In: Proceedings of ACM SIGMOD Conference. (2004) 407–418
3. Fabret, F., Jacobsen, H.A., Llirbat, F., Pereira, J., Ross, K.A., Shasha, D.: Filtering algorithms and implementation for very fast publish/subscribe. In: Proceedings of ACM SIGMOD Conference. (2001) 115–126
4. Ross, K.A.: Conjunctive selection conditions in main memory. In: Proceedings of PODS Conference. (2002) 109–120

# An Efficient and Scalable Management of Ontology

Myung-Jae Park<sup>1</sup>, Jihyun Lee<sup>1</sup>, Chun-Hee Lee<sup>1</sup>, Jiexi Lin<sup>1</sup>,  
Olivier Serres<sup>2</sup>, and Chin-Wan Chung<sup>1</sup>

<sup>1</sup> Korea Advanced Institute of Science and Technology, Korea  
`{jpark, hyunlee, leechun, jesse, chungcw}@islab.kaist.ac.kr`

<sup>2</sup> University of Technology of Belfort-Montbéliard, France  
`olivier.serres@utbm.fr`

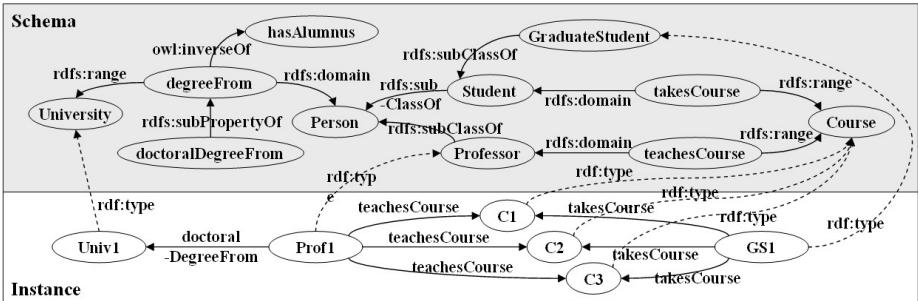
**Abstract.** OWL is a recommended language for publishing and sharing ontologies on the Semantic Web. To manage the ontologies, several OWL data management systems have been proposed. However, the existing systems have limitations of the scalability and the reasoning. In this paper, we propose an OWL data management system, ONTOMS, which stores OWL data into class based relations, performs complete inverseOf, symmetric, and transitive reasoning for instances, and efficiently evaluates OWL-QL queries against ontologies in a relational database.

## 1 Introduction

Web Ontology Language(OWL) [1] is a semantic markup language for publishing and sharing ontologies on the Web. OWL is developed as a vocabulary extension of RDF [2] and RDFS [3] to increase the expressive power of ontology data, which leads OWL as a recommended ontology language for the Semantic Web. OWL data can be represented by a graph like RDF as shown in Fig. 1.

To support the expressive power of OWL data, several OWL reasoners [4, 5, 6] have been proposed. However, those reasoners confronted the scalability issue, due to the use of memory. To overcome this problem, RDBMS based OWL data management systems [7, 8, 9] have been proposed. Since RDBMSs do not support the reasoning ability, those systems cannot obtain complete class and property hierarchies and perform the instance reasoning. As a result, those systems incorporated OWL reasoners to obtain such hierarchies completely. However, due to the scalability drawback of OWL reasoners, instance reasoning is not supported.

To retrieve instances of classes or properties, OWL Query Language(OWL-QL) [10] has been proposed. OWL-QL is based on query patterns, in the form of (property, subject, object). To evaluate query patterns over OWL data stored in RDBMS, a proper relational schema is required. However, existing systems do not incorporate efficiency consideration in designing their relational schemas. Thus, in this paper, we propose ONTOMS, an efficient and scalable ONTOlogy Management System, to efficiently manage large sized OWL data. ONTOMS stores OWL data into a class based relational schema to increase query processing performance. On the average, the query performance of ONTOMS is about



**Fig. 1.** An example of OWL data (a simple university ontology)

90 times better than DLDB [7]. To provide the complete results, ONTOMS supports instance reasoning for inverseOf, symmetric, and transitive properties. To our best knowledge, ONTOMS is the first RDBMS based OWL data management system which supports the complete instance reasoning.

## 2 Related Work

There are several OWL reasoners to manage OWL data. FaCT [5] performs class and property related reasoning only. RACER [4] and Pellet [6] support class and property hierarchy reasoning, as well as instance reasoning.

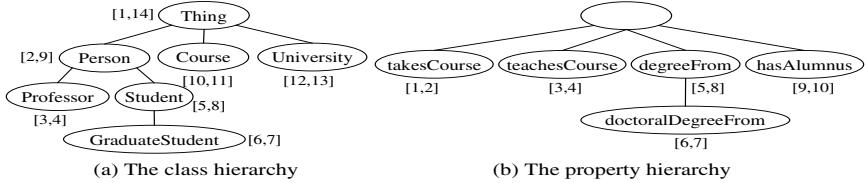
SnoBase [9] stores class and property definitions (e.g., subClassOf and subPropertyOf) into Fact relation. SnoBase also stores every triple (i.e., (subject, property, object)) into Fact relation. To provide reasoning, SnoBase utilizes SQL triggers. However, the runtime depth level of trigger cascading supported in RDBMSs is limited. Also, SnoBase does not support instance reasoning.

Instance Store [8] uses Descriptions relation to store class definitions, Primitives relation to store individuals, and other four relations (Type, Equivalents, Parents and Children) to maintain class hierarchy information. Instance Store uses FaCT or RACER to only obtain class hierarchies. In addition, Instance Store only supports classes without any consideration on properties.

DLDB [7] maintains one class relation for each class and one property relation for each property. For class and property hierarchies, DLDB uses FaCT. However, DLDB does not support any instance reasoning. Thus, DLDB cannot provide complete query results for properties which require instance reasoning.

## 3 OWL Data Storage

The class hierarchy and the property hierarchy are generated from Pellet. To maintain containment relationship information among nodes of each hierarchy, a pair of (start, end) values is assigned to each node according to the node's position, which was originally proposed for XML data [11]. Fig. 2 shows class and property hierarchies for the OWL data in Fig. 1.



**Fig. 2.** Class and Property Hierarchies

ONTOMS generates a class based relational schema, where one relation is created for each class. Each class relation contains associated properties as attributes. Associated properties are the properties that have the class as domains.

To efficiently utilize the property hierarchy, we only retain properties which do not have any super properties (called highest properties) among associated properties. Class relations also have start and end attributes for each highest property. If the highest property has no subproperties, those are omitted.

Student					Course	GraduateStudent	Professor
UID	takesCourse	degreeFrom	degreeFrom_S	degreeFrom_E	UID	_takesCourse	_teachesCourse
Person					University		
UID	degreeFrom	degreeFrom_S	degreeFrom_E	UID	hasAlumnus	GS1	C1
				Univ1		Prof1	C1
					GS1	C2	Prof1
					GS1	C3	Prof1
							C3
GraduateStudent				Professor			
UID	degreeFrom	degreeFrom_S	degreeFrom_E	UID	degreeFrom	degreeFrom_S	degreeFrom_E
GS1				Prof1	Univ1	6	7

**Fig. 3.** Relational Tables in ONTOMS

A number of redundant tuples would be generated for a multi-valued property. The multi-valued property indicates where one instance of property's domain has more than one values. For example, in Fig. 1, Prof1 has three different values (i.e., C1, C2, and C3) for teachesCourse property. As a result, ONTOMS separates multi-valued properties from class relations. A new relation is assigned to each multi-valued property. Fig. 3 shows the relations<sup>4</sup>, generated for the OWL data presented in Fig. 1. Note that ONTOMS stores new tuples generated after performing instance reasoning. The (Univ1, null) tuple in University relation should be updated as (Univ1, Prof1) for the inverseOf property, hasAlumnus.

Note that the translation process of OWL-QL queries to SQL queries for the class based relations can be found in [12].

## 4 Instance Reasoning

OWL defines five types of properties: inverseOf, symmetric, transitive, functional and inverseFunctional properties. Only the first three properties may generate a

---

<sup>1</sup> Internally, ONTOMS assigns a unique label identifier (UID) to each instance.

large number of new facts<sup>2</sup>. Therefore, we focus on reasoning for inverseOf, symmetric and transitive properties (which we will refer to as IST properties). Note that the definitions of the IST properties are given in the OWL Reference [1].

**Definition 1.** *Relation R of a property P is the set of (x,y) in P.*

**Definition 2.** *Let property P be inverseOf property P', R be the relation of P, and R' be the relation of P'. The inverseOf reasoning for P or R is the process of adding (y,x) to R for all (x,y) in R', if (y,x) is not in R.*

**Definition 3.** *Let P be a symmetric property and R be the relation of P. The symmetric reasoning for P or R is the process of adding (y,x) to R for all (x,y) in R, if (y,x) is not in R.*

**Definition 4.** *Let P be a transitive property and R be the relation of P. The transitive reasoning for P or R is the process of computing the transitive closure of R.*

A relation R after inverseOf reasoning is written as  $R^I$ <sup>3</sup>. Similarly, a relation R after symmetric and transitive reasoning is written as  $R^S$  and  $R^T$ , respectively.

In this paper, we propose an IST reasoning algorithm which does not require recursive or iterative processing. The algorithm guarantees that the complete set of new facts can be obtained by performing reasoning only once for each type of property in a certain sequence, i.e., first for inverseOf property, then for symmetric property, and last for transitive property.

**Lemma 1.** *Suppose relation R is inverseOf relation R'. After symmetric reasoning for R and R',  $R^S$  is inverseOf  $R'^S$ .*

**Lemma 2.** *Suppose relation R is inverseOf relation R'. If R is symmetric, then R' is symmetric. If R is transitive, then R' is transitive.*

**Theorem 1.** *Suppose property P is inverseOf property P', P is symmetric and transitive. Let R be the relation of P and R' the relation of P'. By following the sequence <inverseOf reasoning, symmetric reasoning, transitive reasoning> for R and R', the resulting relations of R and R' are inverseOf of each other, symmetric and transitive.*

The proof for Theorem 1 and the algorithm for IST reasoning, which can be found in [12], are not included due to the page limitation.

## 5 Experiments

We implemented ONTOMS using IBM DB2 UDB 8.2. We interfaced DLDB with IBM DB2 since DLDB uses MS-Access. Experiments were performed on 3GHz

<sup>2</sup> Referred to as newly generated instances in Sect. B

<sup>3</sup> Here, we introduce  $R^I$  to indicate the change of R as a result of inverseOf reasoning.

Pentium 4 with 1024MB of main memory. We used Lehigh University Benchmark Data(LUBM)<sup>4</sup>. We generated various sizes of OWL data: 1MB, 5MB, 10MB, 50MB, 100MB, and 500MB. Since LUBM queries (Q1 to Q14) are not sufficient, we added three queries (Q15 to Q17). Detailed information on the query set, which can be found in [2], is not included due to the page limitation.

We compared the total query processing time of ONTOMS and that of DLDB using 17 queries. We show the total query processing time for only 50MB data in Fig. 4 since the shapes of graphs for different sizes are similar.

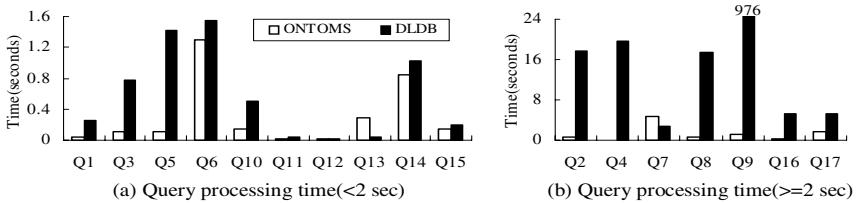


Fig. 4. Query Processing Time for 50MB OWL Data

The number of joins in ONTOMS is less than or equal to that of DLDB. Therefore, in Fig. 4, ONTOMS is better than DLDB for most of 17 queries. However, for Q7 and Q13, ONTOMS is worse than DLDB. Since Q7 and Q13 have values as their subjects, there are just a few bindings satisfying those queries.

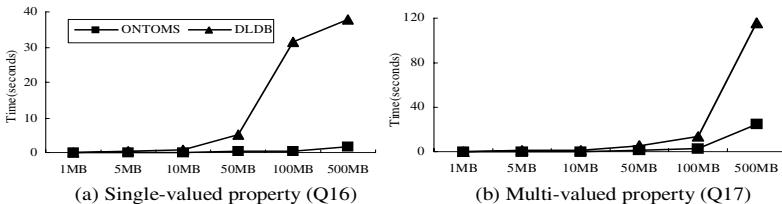


Fig. 5. Query Processing Time over Differently Sized OWL Data

In Fig. 5, all properties of Q16 are single-valued properties while those of Q17 contain multi-valued properties. Thus, the performance gap between ONTOMS and DLDB is much larger in Q16.

Consequently, ONTOMS outperforms DLDB for most queries in spite of its support of instance reasoning. ONTOMS is 90 times faster than DLDB on the average, calculated by averaging performance differences for queries over 1MB, 5MB, 10MB, 50MB, 100MB, and 500MB OWL data.

<sup>4</sup> Available in <http://swat.cse.lehigh.edu/projects/lubm/index.htm>

## 6 Conclusion

In this paper, we proposed ONTOMS, an OWL data management system using an RDBMS. ONTOMS generates the class based relational schema in which a relation is created for each class and contains associated properties as its attributes. To avoid data redundancy, ONTOMS creates class-property relations for multi-valued properties. Thus, this schema is of great advantage to queries having less multi-valued properties. In addition, ONTOMS supports the reasoning on the IST properties and the class and property hierarchies. The experimental results show that ONTOMS outperforms DLDB in the query response time.

**Acknowledgments.** This research was supported by the Ministry of Information and Communication, Korea, under the College Information Technology Research Center Support Program, grant number IITA-2006-C1090-0603-0031.

## References

1. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference. W3C Recommendation, <http://www.w3.org/TR/owl-ref> (Feb. 2004)
2. Manola, F., Miller, E., McBride, B.: RDF Primer. W3C Recommendation, <http://www.w3.org/TR/rdf-primer> (Feb. 2004)
3. Brickley, D., Guha, R.V., McBride, B.: RDF Vocabulary Description Language 1.0: RDF Schema. <http://www.w3.org/TR/rdf-schema> (Feb. 2004)
4. Haarsley, V., Moller, R.: RACER System Description. In: Proc. of 1st International Joint Conference on Automated Reasoning. (June 2001) 701–706
5. Horrocks, I., Sattler, U.: A Tableaux Decision Procedure for SHOIQ. In: Proc. of 19th International Joint Conference on Artificial Intelligence. (Aug. 2005) 448–453
6. Parsia, B., Sirin, E.: Pellet: An OWL DL Reasoner. In: Proc. of 3rd International Semantic Web Conference. (Nov. 2004)
7. Guo, Y., Pan, Z., Heflin, J.: An Evaluation of Knowledge Base Systems for Large OWL Datasets. In: Proc. of 3rd International Semantic Web Conference. (Nov. 2004) 274–288
8. Horrocks, I., Li, L., Turi, D., Bechhofer, S.: The Instance Store: Description Logic Reasoning with Large Numbers of Individuals. In: Proc. of 2004 International Workshop on Description Logic. (June 2004) 31–40
9. Lee, J., Goodwin, R.: Ontology Management for Large-Scale Enterprise Systems. IBM Technical Report, RC23730 (Sep. 2005)
10. Fikes, R., Hayes, P., Horrocks, I.: OWL-QL-A Language for Deductive Query Answering on the Semantic Web. Journal of Web Semantics **2**(1) (Dec. 2004) 19–29
11. Zhang, C., Naughton, J., DeWitt, D., Luo, Q., Lohman, G.: On Supporting Containment Queries in Relational Database Management Systems. In: Proc. of the 2001 ACM SIGMOD Conference. (May 2001) 425–436
12. Park, M.J., Lee, J.H., Lee, C.H., Lin, J., Serres, O., Chung, C.W.: ONTOMS: An Efficient and Scalable Ontology Management System. In: KAIST CS/TR-2005-246. (Dec. 2005)

# Estimating Missing Data in Data Streams\*

Nan Jiang and Le Gruenwald

The University of Oklahoma  
School of Computer Science  
Norman, OK, 73019, USA

{nan\_jiang, ggruenwald}@ou.edu

**Abstract.** Networks of thousands of sensors present a feasible and economic solution to some of our most challenging problems, such as real-time traffic modeling, military sensing and tracking. Many research projects have been conducted by different organizations regarding wireless sensor networks; however, few of them discuss how to estimate missing sensor data. In this research we present a novel data estimation technique based on association rules derived from closed frequent itemsets generated by sensors. Experimental results compared with the existing techniques using real-life sensor data show that closed itemset mining effectively imputes missing values as well as achieves time and space efficiency.

## 1 Introduction

Many research projects have been conducted by different organizations regarding wireless sensor networks; however, few of them discuss how to estimate the sensor data that are missing because they are lost or corrupted or arrive late when being sent from sensors to servers. Traditional methods to handle the situation when data is missing are to ignore them, make sensors to send them again or use some statistical methods to perform the estimation. As we discuss in Section 2, these methods are not specially suited for wireless sensor networks.

In this paper, we propose a data estimation technique using association rule mining on stream data based on closed frequent itemsets (CARM) to discover relationships between sensors and use them to compensate for missing data. Different from other existing techniques [4-6, 10, 12], CARM can discover the relationships between two or more sensors when they have the same or different values. The derived association rules provide complete and non-redundant information; therefore they can improve the estimation accuracy and achieve both time and space efficiency. Furthermore, CARM is an online and incremental algorithm, which is especially beneficial when users have different specified support thresholds in their online queries.

The remainder of this paper is organized as follows. Section 2 describes the data missing problem and reviews the existing data estimation solutions. Section 3 discusses the definitions of terms used in the paper. Section 4 presents the proposed

---

\* This research is partially supported by the NASA grant No. NNG05GA30G and a research grant from the United States Department of Defense.

online data estimation algorithm based on the discovered closed frequent itemsets. Section 5 depicts the performance evaluation comparing the proposed algorithm with the existing techniques using real-life traffic data. Finally, Section 6 concludes the paper.

## 2 Related Works

Many articles have been published to deal with the missing data problem, and a lot of software has been developed based on these methods. Some of the methods totally delete the missing data before analyzing them, like listwise and pairwise deletion [16], while some other methods focus on estimating the missing data based on the available information. The most popular statistical estimation methods include mean substitution, imputation by regression [3], hot deck imputation [7], cold deck imputation, expectation maximization (EM) [10], maximum likelihood [2, 9], multiple imputations [11, 13], and Bayesian analysis [5]. However, a number of problems arise when applying them to sensor networks applications. First, none of the existing statistical methods answers the question that is critical to data stream environments: how many rounds of information should we use in order to get the associated information for the missing data estimation? Second, it is difficult to draw a pool of similar complete cases for a certain round of a certain sensor when it needs to perform the data estimation, which makes some statistical methods difficult to use. Third, since the missing sensor data may or may not be related to all of the available information, using all of the available information to generate the result as described in some of the statistical methods would consume unnecessary time. And fourth, sensor data may or may not Miss At Random (MAR), which makes it unfavorable to use those statistical methods that require the MAR property.

In [6], the authors proposed the WARM (Window Association Rule Mining) algorithm for estimating missing sensor data. WARM uses association rule mining to identify sensors that report the same data for a number of times in a sliding window, called related sensors, and then estimates the missing data from a sensor by using the data reported by its related sensors. WARM has been reported to perform better than the average approach where the average value reported by all sensors in the window is used for estimation. However, there exist some limitations in WARM. First, it is based on 2-frequent itemsets association rule mining, which means it can discover the relationships only between two sensors and ignore the cases where missing values are related with multiple sensors. Second, it finds those relationships only when both sensors report the same value and ignores the cases where missing values can be estimated by the relationships between sensors that report different values.

In view of the above challenges, in this paper we propose a data estimation technique, called CARM (Closed Itemsets based Association Rule Mining), which can derive the most recent association rules between sensors based on the current closed itemsets in the current sliding window. The definition of closed itemsets is given in Section 3 where we describe the notations that are used throughout this paper.

### 3 Definitions

Let  $D = \{d_1, d_2, \dots, d_n\}$  be a set of  $n$  item ids, and  $V = \{v_1, v_2, \dots, v_m\}$  be a set of  $m$  item values. An item  $I$  is a combination of  $D$  and  $V$ , denoted as  $I = D.V$ . For example,  $d_n.v_m$  means that an item with id  $d_n$  has the value  $v_m$ . A subset  $X \subseteq I$  is called an itemset. A  $k$ -subset is called a  $k$ -itemset. Each transaction  $t$  is a set of items in  $I$ . Given a set of transactions  $T$ , the support of an itemset  $X$  is the percentage of transactions that contain  $X$ . A frequent itemset is an itemset the support of which is above or equal to a user-defined support threshold [1].

Let  $T$  and  $X$  be subsets of all the transactions and items appearing in a data stream  $D$ , respectively. The concept of closed itemset is based on the two following functions,  $f$  and  $g$ :  $f(T) = \{i \in I \mid \forall t \in T, i \in t\}$  and  $g(X) = \{t \in D \mid \forall i \in X, i \in t\}$ . Function  $f$  returns the set of itemsets included in all the transactions belonging to  $T$ , while function  $g$  returns the set of transactions containing a given itemset  $X$ . An itemset  $X$  is said to be closed if and only if  $C(X) = f(g(X)) = f \bullet g(X) = X$  where the composite function  $C = f \bullet g$  is called Galois operator or closure operator [14].

From the above discussion, we can see that a closed itemset  $X$  is an itemset the closure  $C(X)$  of which is equal to itself ( $C(X) = X$ ). The closure checking is to check the closure of an itemset  $X$  to see whether or not it is equal to itself, i.e., whether or not it is a closed itemset.

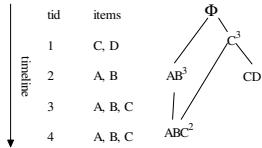
An association rule  $X \rightarrow Y$  ( $s, c$ ) is said to hold if both  $s$  and  $c$  are above or equal to a user-specified minimum support and confidence, respectively, where  $X$  and  $Y$  are sensor readings from different sensors,  $s$  is the percentage of records that contain both  $X$  and  $Y$  in the data stream, called support of the rule, and  $c$  is the percentage of records containing  $X$  that also contain  $Y$ , called the confidence of the rule. The task of mining association rules then is to find all the association rules among the sensors which satisfy both the user-specified minimum support and minimum confidence.

### 4 Data Estimation Algorithm Based on Closed Frequent Itemsets

In this section, we present an online data estimation technique called CARM based on a closed frequent itemsets mining algorithm in data streams that we have proposed recently, called the CFI-Stream [8]. We first briefly describe the CFI-Stream data structure called DIrect Update (DIU) tree that is used to compute online the closed frequent itemsets in data streams. Then we discuss how to estimate the missing data based on the association rules derived from the discovered closed frequent itemsets.

A lexicographical ordered direct update tree is used to maintain the current closed itemsets. Each node in the DIU tree represents a closed itemset. There are  $k$  levels in the DIU tree, where each level  $i$  stores the closed  $i$ -itemsets. The parameter  $k$  is the maximum length of the current closed itemsets. Each node in the DIU tree stores a closed itemset, its current support information, and the links to its immediate parent and children nodes. Fig.1. illustrates the DIU tree after the first four transactions arrive. The support of each node is labeled in the upper right corner of the node itself. The figure shows that currently there are 4 closed itemsets,  $C$ ,  $AB$ ,  $CD$ , and  $ABC$ , in

the DIU tree, and their associated supports are 3, 3, 1, and 2. We assume in this paper that all current closed itemsets are already derived, and based on these closed itemsets, we generate association rules for data estimation. Please refer to [8] for the detail discussion of the update of the DIU tree and the closure checking procedures.



**Fig. 1.** The lexicographical ordered direct update tree

CARM proceeds in the following manner. First, it checks if there are missing values in the current round of sensor readings. If yes, it uses the current round of readings  $X$  that contains the missing items to find out its closure online. If the rules from  $X$  to its immediate upper level supersets satisfy the user specified support and confidence criteria, these upper level supersets are treated as starting points to explore more potential itemsets until CARM estimates all missing sensor data. Following this method, CARM continues to explore and find all closed itemsets that can generate association rules satisfying the users' specified support and confidence criteria. All these closed itemsets are the supersets of the exploration set and have the support and confidence along the path above or equal to the users' specified thresholds.

CARM generates the estimated value based on the rules and selected closed itemsets, which contain item value(s) that are not included in the original readings  $X$ . It weights each rule by its confidence and calculates the summation of these weights multiplied with their associated item values as the final estimated result. These item values can be expected as the missing item values with the support and confidence values equal to or greater than the users' specified thresholds. In this way, CARM takes into consideration all the possible relationships between the sensor readings and weights each possible missing value by the strength (confidence) of each relationship (rule). This enables CARM to produce a final estimated result near the actual sensor value based on all of the previous sensor relationships information. We show the CARM algorithm in Fig. 2, where  $X$  is the itemset in the current round of sensor readings,  $Y$  represents all supersets of  $X$ ,  $\text{Conf}_Y$  represents the strength of the rule from itemset  $X$  to  $Y$ ,  $\text{Support}(X)$  represents  $X$ 's support,  $\text{Closure}(X)$  is the closure of itemset  $X$  in the current transactions,  $\text{Min}(X)$  represents  $X$ 's immediate upper level supersets in the DIU tree,  $C$  represents all closed frequent itemsets,  $S_{(l)}, V_l$  represents the value  $V_l$  of sensor id  $S_{(l)}$ ,  $X_{\text{estimate}}$  represents the returned estimation itemset which contains the sensor ids with missing values in the current round of readings of stream data and their corresponding estimated values,  $S_{\text{specify}}$  represents the user specified support, and  $C_{\text{specify}}$  represents the user specified confidence.

---

```

1 Xestimate=∅;
2 For all (M ⊆ X)
3 confM=1; C_estimate(M, confM, Xestimate)
4 If (Xestimate contains all the missing values)
5 break;
6 End for
7 Procedure C_estimate(X, Confx, Xestimate) {
8 Xnew=∅;
9 If (X=Closure(X))
10 For all (Y>X and Y∈C and Y = min(X))
11 Confy=Confx*Support(Y)/Support(X);
12 Xnew = Xnew ∪ (Y/Xestimate)
13 End for
14 For all (I∈Xnew)
15 For all (Z>X and Z=min(Z))
16 If(I∈Z)
17 ConfI=ConfZ;
18 End for
19 If (Support(I∪X)>Sspecify and ConfI>Cspecify)
20 S(I).VI =S(I).VI +ConfI*VI
21 End for
22 If(Xnew doesn't contain all missing sensor data)
23 For all (X'>X and X'∈C and X'= min(X))
24 Call C_estimate(X', Confx', Xestimate ∪ Xnew)
25 End if
26 Else
27 Xc=Closure(X); Xnew=Xc/X; Confxc=1;
28 If(Support(Xc)> Sspecify)
29 For all (J∈ Xnew)
30 ConfJ = Confxc; S(J).VJ =S(J).VJ +ConfJ*VJ;
31 End if
32 If(Xnew doesn't contain all missing sensor data)
33 Call C_estimate(Xc, Confxc, Xnew)
34 End if
35 End procedure

```

---

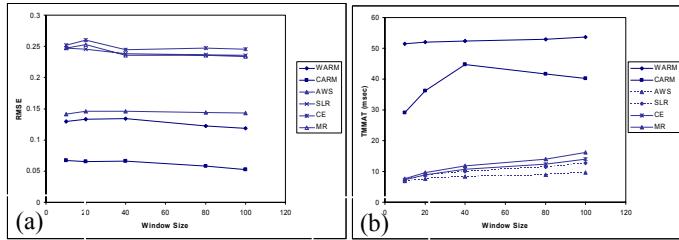
**Fig. 2.** The CARM online data estimation algorithm

## 5 Experimental Evaluations

Several different simulation experiments are conducted comparing CARM with four existing statistical techniques: Average Window Size (AWS), the Simple Linear Regression (SLR), the Curve Regression (CE), and Multiple Regression (MR), and with WARM, a data estimation algorithm in sensor database [6].

As shown in Fig. 3(a), the experiment results show that CARM gives the best estimation accuracy, followed by WARM and AWS. The regression approaches perform worse than WARM, CARM and AWS. The main reason of this might be that

they only based on the regressions between the neighbor sensor readings, while CARM and WARM discover all of the relationships between the existing sensors. CARM provides better estimation accuracy than WARM because the association rules in CARM are derived from a compact and complete set of information, while those in WARM are derived from only the 2-frequent itemsets in the current sliding window.



**Fig. 3.** RMSE and TMMAT for AWS, SLR, CE, MR, WARM and CARM approaches

In terms of TMMAT, which is the time for performing all main memory accesses required for updating the associated data structures and estimating missing values per round of sensor readings, as shown in Fig. 3(b), CARM is outperformed by all other four statistical approaches, but it is still very fast comparing with the cases in which sensors must resend the missing data, and is faster than WARM. The TMMAT of WARM increases slightly when the window size increases since the information in WARM is stored in the cube data structures, and the time needed to process this information increases when the size of the cube increases. For CARM, the TMMAT first increases as the number of transactions increases since the number of closed itemsets that are newly discovered increases.

In terms of Memory Space, CARM is outperformed by all other four statistical approaches, but it still requires far less memory space than that provided in a contemporary computer. The needed memory space in CARM is much lower than that in WARM because the tree data structure used in CARM stores only the condensed closed itemset information while the cube data structures in WARM store the sensor readings of all sensors and the supports of pairs of sensors in the current sliding window.

## 6 Conclusions

In this paper we proposed a novel algorithm, called CARM, to perform data estimation in sensor network databases based on closed itemsets mining in sensor streams. The algorithm offers an online method to derive association rules based on the discovered closed itemsets, and imputes the missing values based on derived association rules. It can discover the relationships between multiple sensors not only when they report the same sensor readings but also when they report different sensor readings. Our performance study shows that CARM is able to estimate missing sensor data online with both time and space efficiency, and greatly improves the estimation accuracy.

## References

1. R. Agrawal, T. Imielinski, A. Swami; Mining Association Rules between Sets of Items in Massive Databases; Int'l Conf. on Management of Data; May 1993.
2. Allison, P. D. Missing data. Thousand Oaks, CA: Sage; 2002.
3. Cool, A. L. A review of methods for dealing with missing data; Annual Meeting of the Southwest Educational Research Association, Dallas, TX. 2000.
4. Dempster, N. Laird, and D. Rubin; Maximum Likelihood from Incomplete Data via the EM Algorithm; Journal of the Royal Statistical Society; 1977.
5. Gelman, J. Carlin, H. Stern, and D. Rubin; Bayesian Data Analysis; Chapman & Hall; 1995.
6. M. Halatchev and L. Gruenwald; Estimating Missing Values in Related Sensor Data Streams; Int'l Conf. on Management of Data; January 2005.
7. Iannacchione, V. G. Weighted sequential hot deck imputation macros. Proceedings of the SAS Users Group International Conference; 1982.
8. N. Jiang and L. Gruenwald, "CFI-Stream: Mining Closed Frequent Itemsets in Data Streams", ACM SIGKDD intl. conf. on knowledge discovery and data mining, 2006.
9. Little, R. J. A., Rubin, D. B. Statistical analysis with missing data; John Wiley and Sons. 1987.
10. G. McLachlan and K. Thriyambakam; The EM Algorithm and Extensions; John Wiley & Sons; 1997.
11. D.Rubin. "Multiple Imputations for Nonresponce in Surveys". John Wiley & Sons; 1987
12. D. Rubin; Multiple Imputations after 18 Years; Journal of the American Statistical Association; 1996.
13. J. Shafer; Model-Based Imputations of Census Short-Form Items; Annual Research Conference, Washington, DC: Bureau of the Census, 1995.
14. R. Taouil, N. Pasquier, Y. Bastide and L. Lakhal; Mining Bases for Association Rules Using Closed Sets; International Conference on Data Engineering; 2000.
15. O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. Altman. "Missing Value Estimation Methods for DNA Microarrays;" Bioinformatics, 17, 2001.
16. Wilkinson & The APA Task Force on Statistical Inference, 1999.

# AB-Index: An Efficient Adaptive Index for Branching XML Queries\*

Bo Zhang<sup>1</sup>, Wei Wang<sup>2</sup>, Xiaoling Wang<sup>1</sup>, and Aoying Zhou<sup>1</sup>

<sup>1</sup> Department of Computer Science and Engineering  
Fudan University, China

{zhangbo, wxling, ayzhou}@fudan.edu.cn

<sup>2</sup> School of Computer Science and Engineering  
University of New South Wales, Australia  
weiw@cse.unsw.edu.au

**Abstract.** Query-adaptive XML indexing has been proposed and shown to be an efficient way to accelerate XML query processing, because it dynamically adapts to the workload. However, existing adaptive index lack of support for branching queries, and also with low efficiency for query processing and adaptation operations. In this paper, we propose a new Adaptive index for Branching queries, which is named as AB-Index. It is designed to support XML path queries with branching predicates. Efficient index construction, query processing, and index adaptation algorithms are proposed for AB-Index. In the experiments, the proposed index is demonstrated to outperform the state-of-the-art approach in the area of adaptive index in terms of query and adaptation efficiencies.

## 1 Introduction

XML has become the standard for data representation and exchange on the Internet. The rapid popularity of XML repositories require systems that can store and query XML data efficiently. Indexing the structure of XML data is an effective way to accelerate XML query processing, because it can greatly reduce the search space. Researchers have proposed various kinds of XML indexes to facilitate the query processing. Among them, adaptive indexes [13,7] are well-known for their high performance as they can adapt their structures to suit the query workload.

However, most of the proposed adaptive indexes can only accommodate a rather limited class of XPath queries efficiently. Little attention has been given to the problem of building an adaptive index for branching queries. For example, APEX [1] is designed to only support *suffix path queries* of the form  $//l_1/l_2/\dots/l_k$  (where  $l_i$  is a tag name) efficiently, and cannot tackle branching queries. Query and update performances of existing adaptive indexes are also insufficient for branching queries.

---

\* This work is partially supported by Sybase Project, NSFC under grants(No.60673137 and 60403019) and National Hi-Tech program under grant 2006AA01Z103.

In this paper, we propose the AB-Index designed for both simple path queries and branching queries. The basis of AB-index is F&B index [2], thus we can index and manipulate a group of “similar” nodes, which are nodes in F&B index. We propose efficient algorithms to construct and update the index, as well as evaluate queries using the index. Our experiment results show that the proposed index significantly outperforms previous approach in both query and adaptation efficiencies.

## 2 The AB-Index

### 2.1 Overview of the AB-Index

An F&B index[2] for the XML data is a partition of nodes according to their incoming and outgoing paths, such that it can answer all the branching queries. However, an F&B index is query-independent and could be *over-refined* and thus *sub-optimal* for a given query workload. For example, answering the query  $//e$  using the F&B index in Figure 1 requires a traversal of the *complete* F&B index. The basic idea of our AB-Index is to group F&B index nodes according to the frequent queries in a given workload, such that the frequent queries can be efficiently answered.

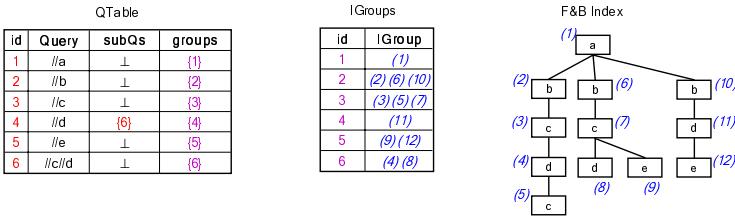


Fig. 1. Example AB-Index (' $\perp$ ' denotes NULL pointer)

The AB-Index consists of three parts: an F&B index for the XML data, *IGroups* and *QTable*. Figure 1 shows the example AB-Index adapted for the frequent query  $//c//d$ .<sup>1</sup>

Each entry of *IGroups* keeps a group of F&B index nodes. Each group belongs to one query as part of the query result. One property is that all the F&B index nodes within the same *IGroup* will either be accessed together for a query or none of them will be accessed.

The *QTable* records a list of frequent queries. Each entry of the *QTable* consists of three fields: *Query*, *subQs* and *groups*. The *Query* field keeps the queries. The *subQs* field is a list of child queries that are directly contained by the current query. This design eliminates the data redundancy problem and also facilitates containment checking [4] which is frequently used in query processing and adaptation processes. The *groups* field is a list of pointers, each pointing to one group in the *IGroup* table.

<sup>1</sup> The rest queries (i.e.,  $//tag$ ) are added in the initialization phase.

The above structure enable us to keep the containment relationship between frequent queries, and it is easy to get the result for a frequent query. Let  $Result(Q)$  be the result of a query in the form of a list of its corresponding F&B index nodes. Our design of the index has the following property for every query in the  $QTable$ :

$$Result(Q) = (\cup_{Q_i \in Q.subQs} Result(Q_i)) \cup Q.groups \quad (1)$$

## 2.2 Initializing the AB-Index

The initialization phase is similar to that of the APEX [4] in that we add a list of hypothetical queries in the form of  $//tag$  to the index. The purpose of inserting such queries is to ensure that there *always* exists a query in the AB-Index that contains any new queries. The algorithm to build the initial AB-Index is rather straight-forward: (a) firstly we build the full F&B index; (b) secondly we combine all the F&B index nodes with the same tag name into a group and put them into the  $IGroup$  table; (c) finally, we insert all the query  $//tag$  into the  $QTable$ , and link them to the corresponding  $IGroup$  entries.

## 2.3 Adapting the AB-Index

The AB-Index is a workload-aware index that can adapt to new frequent queries and remove infrequent queries from its data structure, based on the information collected by a built-in query statistics module. The update process is supported by two basic operations: *inserting* queries, and *deleting* queries.

**Insert New Frequent Queries.** The main task here is to adjust the containment relationship for queries in  $QTable$  to accommodate a new query  $Q_n$ . We first find a minimal query,  $Q_p$ , that contains  $Q_n$ . To judge the containment relationship, we first convert the queries into PatternTrees [4] and then use the containment judgement method in [4]. Next, we need to insert  $Q_n$  as a child query of  $Q_p$ , and adjust the  $subQs$  and  $groups$  fields in  $QTable$  of existing queries as we need to ensure there is no duplicates. Consider inserting a new query  $Q_n$  under another query  $Q_p$ . Denote the sibling queries of  $Q_p$  is  $sibling(Q_p)$  (i.e., the queries correspond to the sibling nodes of  $Q_n$  in the AB-Index), then it is sufficient to adjust queries that are descendants of  $Q_p$  or any query in  $sibling(Q_p)$ . The update algorithm thus updates the AB-Index in a top-down, recursive manner for all the affected queries.

**Delete Old Infrequent Queries.** When a query cease to be frequent among the recent  $N$  queries, we have to delete it from the AB-Index(the query is called as  $Q_d$ ). The main task is to adjust the  $subQs$  and  $groups$  in  $Q_d$ . The algorithm works by first finding all the parent queries that contains  $Q_d$  in their  $subQs$  fields,

then copying  $Q_d.\text{sub}Qs$  and  $Q_d.\text{groups}$  to the result of  $Q_d$ 's parent queries, and removing  $Q_d$  at last.

## 2.4 Query Processing over the AB-Index

Given a new query  $Q_n$ , if it is identical to one of the existing query in the AB-Index, it can be easily answered by fetching its result in a recursive manner according to [10]. The AB-Index, however, may efficiently answer query that does not *exactly* match the frequent queries, thus significantly boosts the performance of the system. We introduce two such approaches in the following.

**Utilize Matching Queries.** We use the following example to illustrate the intuition behind the “match” concept and its query processing method.

**Example 1.** Consider  $Q_n = //a[./b]/c/d/e$  and  $Q_m = //a/c/d$ . The prefixes of depth 3 for  $Q_n$  and  $Q_m$  are  $//a[./b]/c/d$  and  $//a/c/d$ , respectively. Although the prefixes are not identical, the former is contained by the latter. The suffix of the query  $Q_n$  queries is  $./e$ . Our observation is that we can answer  $Q_n$  with  $Q_m$  by refining  $Q_m$ 's result from an upward composing query<sup>2</sup>  $Q_c^{up} = \backslash\backslash[./b]$  (where  $\backslash$  denotes the parent axis), and then evaluate a downward composing query  $Q_c^{down} = ./e$  for the qualified results after the previous step.

**Utilizing Query Containment Relationships.** The second approach is based on the idea that a frequent query,  $Q_p$  might contain a superset of query result of a new query  $Q_n$ .

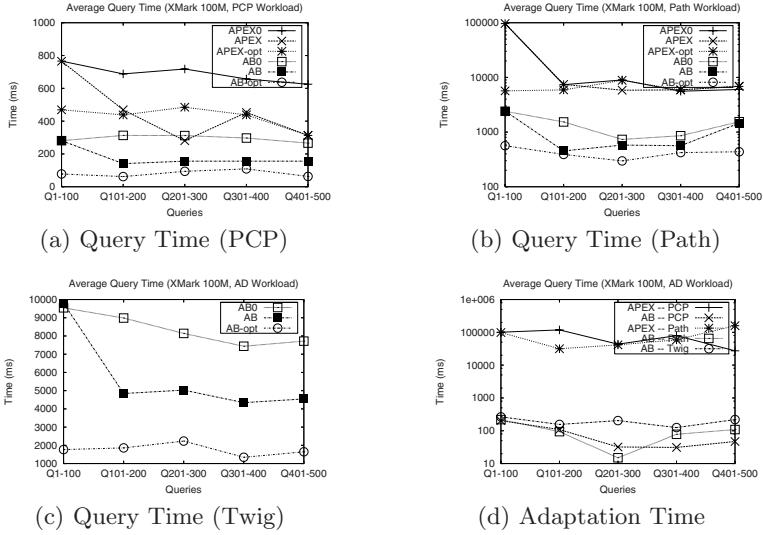
**Example 2.** Consider  $Q_n = //a[./b]/c/d/e$  and  $Q_p = //a//e$ . Since  $Q_p \supseteq Q_n$ , we can obtain the result of  $Q_n$  by validating  $Q_p$ 's result against  $Q_n$ . The validation is essentially evaluating an upward composing query  $Q_c^{up} = \backslash d\backslash c\backslash a[./b]$ . In addition, if we have another query  $Q'_p = //a[./b]/c//e$ , since  $Q_p \supseteq Q'_p \supseteq Q_n$ , we should answer  $Q_n$  using the minimal query that contains  $Q_n$ , which is  $Q'_p$ .

## 3 Experimental Evaluation

In this section, we report experimental results conducted on AB-Index (abbreviated as **AB**) in comparison with APEX index [11]. We implement the construction, update and evaluation algorithms of APEX. All algorithms are implemented in Java 1.4 and ran on a PC of 3.2GHz CPU, 2GB memory, and 80G hard disk. We measure the query processing time and index update time. The dataset used is 100M XMark [5] data.

Three kinds of query workload, **PCP** (for path queries with only / axes), **Path** (general path queries), and **Twig** (general twig queries) are created by adapting the query generator in YFilter [8]. Note that APEX cannot support Twig workload. All workloads have 500 queries and are divided into 5 batches.

<sup>2</sup> Given two queries  $X$  and  $Y$ , a composing query of  $X$  to  $Y$  is a query  $C$  such that  $C \circ X = Y$  [6].

**Fig. 2.** Experiment Results

**Query Performance.** Query performances of APEX and AB is shown in Figures 2(a), 2(b) and 2(c). Note that for each kind of index, we apply three variations: (a) Index0: we construct a static initial index; (b) Index: the index is adapted to the *past* workload after each batch of queries; (c) Index-opt: the index is adapted to the *next* batch of workload. Hence, Index0 gives the baseline performance of a static index, and Index-opt is an offline optimal adaptive index. Measuring the performance of Index0 and Index-opt gives us a better picture of how Index performs.

The following observations can be made:

- The adaptive versions of the indexes outperform their static versions (for both APEX and AB). This is expected as adaptive index can reduce the query processing time of frequent queries.
- The performances of both indexes are close to their optimal versions. Fluctuations are mainly due to the changes of the frequent query distributions as we adapt indexes to the current batch of queries, which might be sub-optimal for the next batch of queries. In addition, the average query time drops quickly after the first adaptation and remains fairly stable in the subsequent updates. This shows that both indexes can quickly adapt to the query workload and start to perform quite well even if there are small fluctuations in the query workload.
- AB outperforms APEX, especially for the **Path** workload. This is because APEX cannot deal with queries in the **Path** workload directly, and have to rewrite queries into several **PCP** queries and then execute each query respectively. It is the rewriting process which depresses the performance of APEX. Also, AB-Index outperforms APEX because the query processing

method for non-frequent queries in AB-Index boost the query performance greatly.

- For the Twig queries, AB's performance is between AB0 and AB-opt. This also shows that if the frequent query distribution is stable, AB can approach the performance of AB-opt, which greatly reduces the query processing time for frequent branching queries.

**Update Performance.** We show the efficiencies of adapting APEX and AB for different workloads in Figure 2(d). We can draw the following observations:

- AB significantly outperforms APEX in update performance. This is mainly because APEX need to traverse the *entire* XML data tree for each and every update. In contrast, our AB-Index uses the query hierarchy to narrow down the update scope and rearrange F&B index nodes instead of its extents.
- As expected, the average adaptation time for the three workloads is generally ordered as PCP < Path < Twig due to the increasing complexities in the update routines.
- The adaptation time for AB is only a small fraction of the query time. So even if we include the cost of update into the AB's query time, it still outperforms both the static AB (AB0) and APEX.

## 4 Conclusions

In this paper, we introduce the AB-Index, which is a workload-adaptive index for XML branching queries. The AB-Index organizes frequently occurring queries and their results in the query workload as in a hierarchical and non-redundant way. Efficient index construction, query processing and adaptation algorithms have been proposed. The effectiveness of the proposed index has been demonstrated in the experiment.

## References

1. C. Chung, J. Min, and K. Shim. APEX: An Adaptive Path Index For XML Data. In *SIGMOD 2002*: 121-132.
2. R. Kaushik, P. Bohannon, J. F. Naughton and H. F. Korth. Covering Indexes for Branching Path Queries. In *SIGMOD 2002*: 133-144.
3. H. He and J. Yang. Multiresolution Indexing of XML for Frequent Queries. In *ICDE 2004*: 683-694.
4. G. Miklau and D. Suciu. Containment and Equivalence for an XPath Fragment. In *PODS 2002*: 65-76.
5. XMark Data Set. <http://monetdb.cwi.nl/xml>
6. B. Mandhani and D. Suciu. Query Caching and View Selection for XML Databases. In *VLDB 2005*: 469-480.
7. Damien K. Fisher and Raymond K. Wong. Adaptively indexing dynamic XML. In: *DASFAA 2006*:233-234
8. Y.Diao, P.Fischer, M.Franklin, and R.To. Yfilter: Efficient and scalable filtering of XML documents. In:*ICDE 2002*:341

# Semantic XPath Query Transformation: Opportunities and Performance

Dung Xuan Thi Le<sup>1</sup>, Stephane Bressan<sup>2</sup>, David Taniar<sup>3</sup>, and Wenny Rahayu<sup>1</sup>

<sup>1</sup> La Trobe University, Australia

{dx1le, w.rahayu}@cs.latrobe.edu.au

<sup>2</sup> National University of Singapore

steph@nus.edu.sg

<sup>3</sup> Monash University, Australia

David.Taniar@infotech.monash.edu.au

**Abstract.** In this paper we identify the opportunities for the semantic transformation of XPath queries using the structural and explicit semantics defined in an XML schema. Our classification of transformation is the *semantic path expression* where a path can be semantically *contracted, expanded or complemented*. Among several applications of such transformations, an obvious one is the semantic optimization of XPath queries. The transformation is likely to result in an improved response time for a given system. We empirically evaluate the gain or loss of performance of the identified transformations with two representative off-the-shelf XML data management systems and XPath query processors.

**Keywords:** XML, XPath, Query Processing, Semantic XML Query Optimization.

## 1 Introduction

Semantic query optimization is the process of rewriting, under the knowledge of some integrity constraints, a query into a semantic equivalent one that can be processed more efficiently [3,4,5,7]. The common availability of structural and explicit constraints in XML Schema coupled with XML data renews the interest for the study of semantic query optimization for XML query languages such as XPath and XQuery as well as the optimization of programs in XML languages such as XSL.

Semantic query optimization for XML data has been discussed [9, 10, 12] earlier. However, the problem of these proposed solutions is that they have completely ignored the usefulness of the unique path locations which can be easily defined and traced in the XML Schema. The existing work explicitly focuses on the early possible binding variables before it proceeds to the path processing and checks the presence of three constraints including Occurrence, Inclusive and Exclusive.

Ontology [12] has foreseen an optimization opportunity from an Object-Oriented design perspective, which has explicitly excluded the obvious important structure of XPath expression such as location paths.

In this paper we present a simple typology of the opportunities for the semantic query rewriting of XPath queries into their equivalent ones using the unique path location constraints specified in the XML Schema Definition (XSD). We evaluate the practical potential of these semantic path transformations by comparatively assessing the performance of the workload of XPath queries and their transformations. We use two system representatives of the state of the art XML database management and query processing systems: native XML database system (quoted as XMS) and XML-enabled database system (quoted as XDB).

## 2 Related Work

Semantic query optimization has been extensively studied for relational and deductive databases [2, 4, 5, 11]. The seminal work [2] proposes a typology of rewriting including literal insertion; literal elimination; range modification (modification of a condition); and queries that can be answered without accessing the database (typically when a contradiction is exposed in the query, which, consequently, denotes an empty answer).

Earlier work in semantic query optimization for XML data includes a query tree technique [9, 10] for representing the structural query pattern of an XQuery, and using some primitive constraint definitions in the schema to assist with the derivation of semantic rules. Their goal is to reduce the unnecessary computation for minimizing the buffer size. This technique has a problem in handling the detection of the descendants in a given path.

XPath in [15] shows the contents such as descendent edges ‘//’, wildcard selection, and branching are decidable. Simple path expressions [14] excludes ‘//’ in order to minimize the complexity of queries and uses restricted fragment of XPath (\*,/,[],[]), which can also be processed in polynomial time by pruning redundant nodes.

The rather obvious range modification and inaccessibility have been introduced as semantic optimization techniques for relational, deductive and object databases, are still applicable to XPath and XML schema. Literal insertion and elimination are reciprocal thus it is difficult to find a deterministic optimization algorithm as there is always the possibility of endlessly applying literal insertion and literal elimination. Whereas range modification and transformations that shows the query can be answered without accessing the database are related to value constraints (as opposed to structural constraints).

In order to see the effectiveness of the full rewriting, our work would cover a range of transformation contraction, expansion and complement to answer the query without accessing the database. In particular with the path expression context, for this paper, we apply semantic expansion, contraction and complement to achieve optimization in accessing data using semantic path locations.

## 3 A Typology of Semantic XPath Query Rewriting Opportunities

In this section we present our typology of semantic query transformation opportunities by formulating a set of definitions.

Our work is different from the existing work in that we concentrate on the unique location path constraints defined in the schema to find opportunities to rewrite XPath queries with a consideration of one or more transformation rules including (i) *Semantic Path Expansion*; (ii) *Semantic Path Contraction*; or (iii) *Semantic Path Complement*. We utilize the **key** and **keyref** constraints to locate the unique path location and apply them in our rewiring. XPath is formed by one or more nodes  $n$  where  $\forall n \in N$  and one or more operators  $p_o$ .  $N$  is a sequence of ordered nodes  $n$  where  $N = \{n_1, n_2, n_3 \dots n_i\}$  and  $P_o = \{p_{o1}, p_{o2}, \dots p_{on}\} = \{/ , * , // \dots\}$ . While the “ $*$ ” represents an unknown node in a given path, the “ $//$ ” allows a descendant from any specific node. Operator ‘..’ is the parent of the current node. These operators give opportunities for a derivation of semantic path transformations.

- a. **Semantic Path Expansion.** An XPath is semantically expanded.  $sExpand()$  is a function.

**Definition 1:** Let  $P_e$  be a user XPath and  $P_u$  be a unique path pre-defined in the schema.  $P_e = \langle \exists n \in N, \exists p_o \in P_o \rangle$ ;  $P_u = \langle N, '/' \rangle$ .  $P_u \leftarrow sExpand(P_e)$  and  $P_u \subseteq P_e$ , and target node  $n_m$  occurred only once in the XML schema document where  $m \geq 1$ .

For example  $P_e = /a/b/*/e$  and  $P_u = /a/b/c/e$ . Let  $R_1$  be the result set of  $P_u$  and  $R_2$  is the result set of  $P_e$ .  $P_u \equiv P_e$  iff ‘ $*$ ’ is proved to be  $c, e$  where  $e$  does not re-occur in the XML schema document  $T$ , and  $R_1 \equiv R_2$ .

- b. **Semantic Path Contraction.** An XPath  $P_e$  is semantically contracted.  $sContract()$  is a function.

**Definition 2:** Let  $P_e$  be a user XPath and  $P_u = \{p_{u1}, p_{u2} \dots p_{ui}\}$  be a collection of unique paths pre-defined in the schema.  $P_e = \langle \exists n \in N, \exists p_o \in P_o \rangle$ ;  $p_{ui} = \langle N, '/' \rangle$  and  $sContract(p_{ui}) = \langle n_m, '/' \rangle$ ;  $sContract(P_u) \leftarrow sContract(P_e)$  iff  $P_u \subseteq P_e$  and target node  $n_m$  re-occurs in XML schema document where  $i \leq m$ .

For example  $P_e = /a/*//e$  and  $P_u = \{p_{u1}, p_{u2}\}$  where  $p_{u1} = /a/b/c/e, p_{u2} = /a/b/d/e$ . Let  $R_1$  be the result set of  $P_e$ ,  $R_2$  is the result set of  $p_{u1} \cup p_{u2}$ .  $P_u = //e$  is the semantic path contraction of  $P_e$  iff  $e$  occurs in all  $p_{ui}$  where  $i \geq 1$ ,  $e$  is a target node of  $P_e$  and  $P_u$  and  $R_1 \equiv R_2$ .

- c. **Semantic Path Complement.** An XPath  $P_e$  semantically complementary.  $sComplement()$  is function.

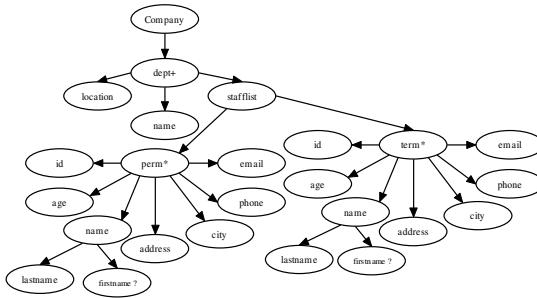
**Definition 3:** Let  $P_e$  be a user XPath and  $P_u = \{p_{u1}, p_{u2} \dots p_{ui}\}$  be collection of unique paths pre-defined in the schema.  $P_e = \langle \exists n \in N, \exists p_o \in P_o \rangle$ ;  $p_{ui} = \langle N, '/' \rangle$  where  $i \geq 1$ .  $\exists p_u \in P_u \leftarrow sComplement(P_e)$  iff  $P_u \subseteq P_e$ , and target node  $n_i$  occurs only once in the XML schema.

For example  $P_e = //b/*/e/..f$  and  $P_u = \{p_{u1}, p_{u2}\}$  where  $p_{u1} = /a/b/c/f, p_{u2} = /a/b/d/f$ . Let  $R_1$  be the result set of  $P_e$ ,  $R_2$  is the result set of  $p_{u2}$ .  $p_{u2}$  is the semantic path complement of  $P_e$  iff  $R_1 \equiv R_2$  where  $i \geq 1$ .

## 4 Empirical Performance Evaluation

### 4.1 Performance Evaluation Strategy and Experimental Set-Up

We now evaluate the potential for optimization created by these proposed *Semantic XPath Transformations*. For each query and its transformed counter-part in our workload, we compare their performance. There exist essentially two practical approaches to the management of XML data and to the processing of queries to native XML database system and XML-enabled database system. For this series of experiments we adopt the schema of Fig 1.



**Fig. 1.** Company Data Schema

**Table 1.** Queries and their Semantic Transformation

Query (n)	XPath (Qn)	Transformation (QnR)
1	<code>/*//ages</code>	<code>/ages</code>
2	<code>//department/*/perm</code>	<code>/company/department/staffList/perm</code>
3	<code>//department/staffList/perm/..//contract</code>	<code>/company/department/staffList/contract</code>
4	<code>/*//perm/ages</code>	<code>/department/staffList/perm/ages</code>

We use three data sets (compliant with this schema) of varying sizes: 15, 25 and 40 mega bytes. The work load, Table 1, is constituted of the queries and of their transformed equivalent queries as given in Table 1. In each result graph, the original queries are labeled as Qn and their transformed counterpart labeled as QnR, where  $n$  is the sequence of the query ( $1 \leq n \leq 4$ ). We measure, for each query and its transformation, their performance on both a native XML database system (XMS) and an XML-enabled database system (XDB). The experiments are performed on a PC AMD Athlon 64 3200+, 2300 MHz 1.0 GB of RAM. The PC is disconnected from the network.

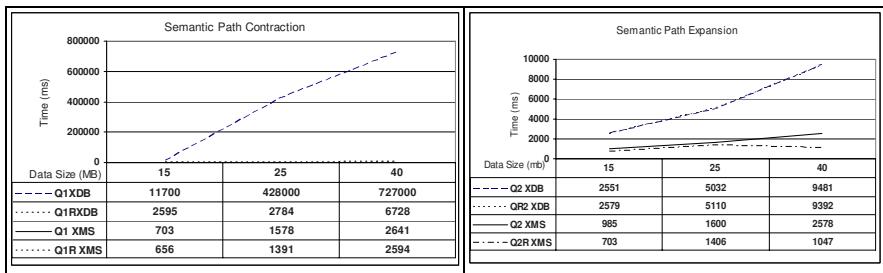
### 4.2 Results

Fig 2 shows the query response time increased as the size of data is increased. Our *semantic path contraction* query is a very useful and effective for an XPath that starts

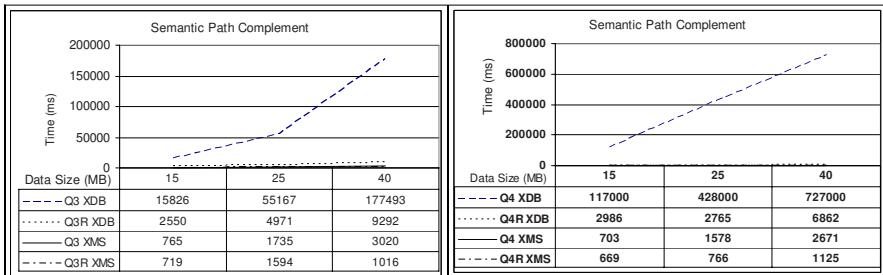
with a wildcard selection “\*” followed by a descendant “//”. The result shows a reduction of response time about 90% for semantic XPath queries in the XDB and nearly 10% for those in XMS.

The result of semantic query applied the *semantic path expansion* transformation in Q2, has shown a very confident reduction of response time between 15%. As for the XDB, our *semantic path expansion* achieves a slight 4% reduction of response time for a single node XPath attribute type expanded to a full XPath.

In Fig 3, the results show that our *semantic path complement* transformation in the commercial mainstream XDB is a significant optimization since the response time of the semantic query shows a reduction almost 95% for all data size in both Q3 and Q4. In the commercial native XMS, our *semantic path complement* transformation gives a significant optimization and even better when the data grows in size, as evidently shown in the results for both Q3 and Q4, that it is mostly 70% of reduction of response time for the large data size experimentation.



**Fig. 2. Queries and Rewriting Queries**



**Fig. 3. Queries and Rewriting Queries Using Path Complement**

The performance of queries in the commercial mainstream XMS is generally better than the performance of queries in XDB. We currently study the main cause of the differences in performance, probably investigating the storing process of XML data in the two mainstreams.

## 5 Conclusion

We have identified and classified a family of possible semantic transformations of XPath queries into equivalent queries using explicit semantic path location constraints available in the schema. We have quantified empirically the potential for optimization using these proposed semantic transformations in two systems representative of the existing options for the management of XML data and for the processing of queries in XML databases. The results highlight significant opportunities for optimization, which although comparatively different, exist in both systems. Our ongoing work focuses on fine tuning the algorithm, and evaluating its cost.

## References

1. Amer-Yahia, S., Cho, S., Lakshmanan, V., Srivastava, D.: Minimization of Tree Pattern Queries. In Proceedings of the ACM SIGMOD Conference on Management of Data (2001) 497 – 508.
2. Charkravarthy, U. S., Grant, J., Minker, J.: Logic-Based Approach to Semantic Query Optimization. In ACM Transactions on Database Systems. Vol. 15, No. 2, (1990) 162-207.
3. Deutsch, A., Popa, L., and Tannen, V.: Query Reformulation with Constraints. SIGMOD Rec. 35, 1 (Mar. 2006), 65-73.
4. Hammer, M., Jdondik, S. B: Knowledge-based processing. In Proceedings of the 6<sup>th</sup> Very Large Databases (VLDB) Conference (Montreal, 1980) IEEE, 137-146.
5. King, J.: Quist: A system for semantic query optimization in relational databases. In Very Large Database (VLDB), IEEE Computer Society (1981) 510-517.
6. Koch, C., Scherzinger, S., Schewikardt et al.: Flux Query: An Optimizing XQuery Processor for Streaming XML Data. In Proceedings of the 30<sup>th</sup> Very Large Data Bases (VLDB) Conference. Toronto, Canada. (2004) 228-239.
7. Shenoy, S. T . Ozsoyoglu, Z. M.: Design and Implementation of a Semantic Query Optimizer. IEEE Transactions on Knowledge and Data Engineering (1987), Vol. 1, No. 3, 344 -361.
8. Su, H., Jian, J., Rundensteiner, E.: Raindrop : A Uniform and Layered Algebraic Framework for XQueries on XML Streams. In International Conference on Information and Knowledge Management (CIKM), New Orleans, Louisiana, USA. ACM. (2005) 279 – 286
9. Su, H., Murali, M., Rundensteiner, E.: Semantic Query Optimization in an Automata Algebra Combined XQuery Engine over XML Streams. In Proceedings of the 30<sup>th</sup> Very Large Data Bases (VLDB) Conference. Toronto, Canada (2004) 1293-1296
10. Su, H., Rundensteiner, E., Mani, M.: Semantic Query Optimization for XQuery over XML Streams. Proceedings of the 31<sup>st</sup> International Conference on Very Large Data Bases (VLDB) Trondheim, Norway (2005) 277-282
11. Sun, J., Zhu, Q.: Probability Based Semantic Query Transformation. In IEEE International Conference on +Systems, Man and Cybernetics (2002) (SMC) Volume 1, 609 – 611.
12. Sun, W., Liu, D.: Using Ontologies for Semantic Query Optimization of XML Databases. Knowledge Discovery from XML Documents: First International Workshop on Knowledge Discovery from XML Documents (KDXD), LNCS (2006) 64 -73

13. Wang,, G., Liu, M., Yu, J.: Effective Schema-Based XML Query Optimization Techniques. In Proceedings of the Seventh International Database Engineering and Application Symposium (IDEAS) (2003). IEEE, 1-6
14. Wood, P.: Minimizing Simple XPath Expression. In the Proceedings of the 4<sup>th</sup> International Workshop on Web and Databases (WebDB), Madison, Wisconsin (2002) 13 - 18.
15. Wood, P.: Containment for XPath Fragments under DTD Constraints. In the Proceedings of the 9<sup>th</sup> International Conference on Database Theory (ICDT), (2003) 300-314.

# TGV: A Tree Graph View for Modeling Untyped XQuery

Nicolas Travers<sup>1</sup>, Tuyêt Trâm Dang Ngoc<sup>2</sup>, and Tianxiao Liu<sup>3</sup>

<sup>1</sup> PRISM Laboratory - University of Versailles, France

[Nicolas.Travers@prism.uvsq.fr](mailto:Nicolas.Travers@prism.uvsq.fr)

<sup>2</sup> ETIS Laboratory - University of Cergy-Pontoise, France

[Tuyet-Tram.Dang-Ngoc@u-cergy.fr](mailto:Tuyet-Tram.Dang-Ngoc@u-cergy.fr)

<sup>3</sup> ETIS Laboratory - University of Cergy-Pontoise & XCalia S.A, France

[Tianxiao.Liu@u-cergy.fr](mailto:Tianxiao.Liu@u-cergy.fr)

**Abstract.** Tree Pattern Queries [7,6] are now well admitted for modeling parts of XML Queries. Actual works only focus on a small subpart of XQuery specifications and are not well adapted for evaluation in a distributed heterogeneous environment.

In this paper, we propose the TGV (Tree Graph View) model for XQuery processing. The TGV model extends the Tree Pattern representation in order to make it intuitive, has support for full untyped-XQuery queries, and for optimization and evaluation. Several types of Tree Pattern are manipulated to handle all XQuery requirements. Links between Tree Patterns are called hyperlinks in order to apply transformations on results.

The TGV<sup>1</sup> has been implemented in a mediator system called XLive.

**Keywords:** XQuery evaluation, TGV, Extensible optimization, Cost model.

## 1 Introduction

XQuery [9] has proved to be an expressive and powerful query language to query XML data both on structure and content, and to make transformation on data. In addition, its query functionalities come from both the database community (filtering, join, selection, aggregation), and the text community (supporting and defining function as text search). However, the complexity of the XQuery language makes its evaluation very difficult. To alleviate this problem, most of the systems support only a limited subset of the XQuery language.

XQuery expressions require a logical model to be manipulated, optimized and then evaluated. [1] introduced the TPQ model that expresses a single *FWR* query by a Pattern Tree and a formula. Then, [2] proposes GTPs that generalizes TPQs with several Pattern Trees, the formula contains all the operations. The representation is quite intuitive and acts as a template for the data source.

<sup>1</sup> The XLlive system and TGV is supported by the ACI Semweb project. TGV annotations and cost models are supported by the ANR PADAWAN project.

However, GTPs do not capture well all the expressiveness of XQuery, cannot handle mediation problems, and do not support extensible optimization.

We design a model called TGV which provides the following features: (a) It integrates the whole functionalities of XQuery (collection, XPath, predicate, aggregate, conditional part, etc.) (b) It uses an intuitive representation that provides a global visualization of the request in a mediation context. (c) It provides a support for optimization and a support for data evaluation.

In this paper we describe the TGV model for evaluating XQuery on heterogeneous distributed sources. This article is organized as follows. The next section introduces the TGV structure that we had defined for modeling XQuery in a practical way. Finally section 3 concludes with the TGV framework.

## 2 XQuery Modeling

XQuery modeling is a difficult goal since the language provides lots of functionalities. And it is all the more difficult as it needs to match mediation requirements (data localization on sources, heterogeneous sources capabilities).

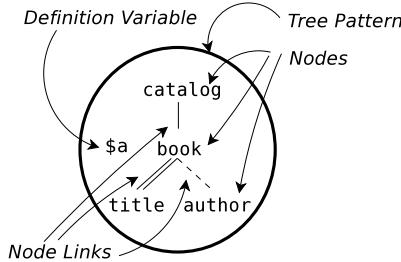
Tree Pattern matching becomes usual in XQuery modeling, trees contain nodes and links, a formula constraints the tree pattern on tags, attributes and contents. Since GTP, it contains joins, nesting, aggregates and optionality.

However, GTP does not handle distributed queries requirements. In fact, data sources are not included, nor XML result constructor, nor views and query on views modeling, nor *Let* and functions and tags, relations and constraints are embedded in a boolean formula difficult to read. Moreover, there is no support for additional information useful for optimization. Thus, this model requires some extensions and adaptations to be the core of a distributed query-processing algorithm in a mediator. We propose the TGV (Tree Graph View) model.

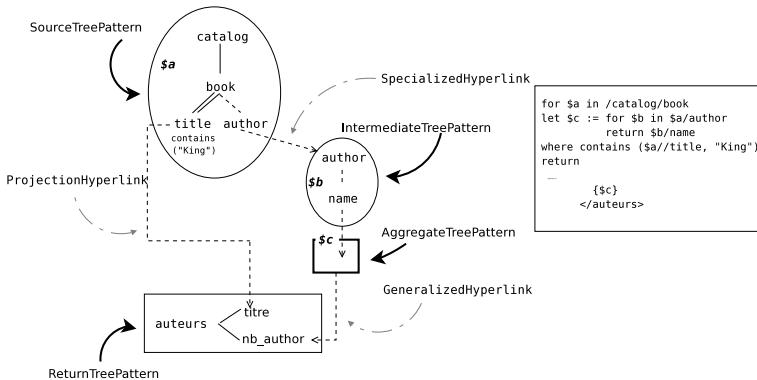
Let see all characteristics of the TGV model. First, we introduce *TreePatterns* which are the XML document filters, and specific structures adapted to XQuery requirements. Then, *Constraints* are added to this model to integrate general filters, which can be attached to any type of the model. To complete this model, *Hyperlinks* are introduced to link together preceding structures. A Tree Graph View is composed of all this structure to model a complete XQuery query.

**Tree Pattern.** A *Tree Pattern* is a tree with different tags an XML document must match with. This template is a set of *XPaths* extracted from the XQuery query. *TreePatterns* are composed of *Nodes* from a label, *NodeLinks* that represent axis between *Nodes* (child, descendant, etc.), and a mandatory/optional status. A Pattern Tree is illustrated on Figure 1. Specific Tree Patterns are integrated to model each characteristic of XQuery illustrated in figure 2.

- A *Source Tree Pattern* (STP) is defined by a targeted document and a root path. It corresponds to a **for** declaration on a targeted XML document with a specific root path, that defines the set of trees to work with.
- An *Intermediate Tree Pattern* (ITP) specializes a previous TreePattern on a specific Node. It corresponds to a **for** declaration with a new path from a



**Fig. 1.** Example of a Node and of a NodeLink in a TreePattern



**Fig. 2.** Four types of Tree Patterns and three types of Hyperlinks

previous variable that specializes an element by creating a new set of trees. Thus, it creates a Tree Pattern that defines a new domain.

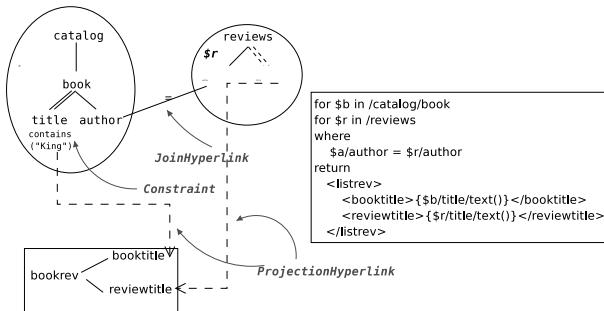
- A *Return Tree Pattern* (RTP) defines the result construction of an XML document. It corresponds to the `return` clause of an XQuery query, which builds the main XML resulting document. Nodes is identified by tags, attributes with a "@", quoted texts, and required XPaths.
- An *Aggregate Tree Pattern* (ATP) builds a temporary result set. It corresponds to a `let` clause that defines a treatment on a set of trees. By canonization rules, nested queries and aggregate functions are defined in those clauses, so they build a temporary result set.

**Constraints.** In XQuery queries, constraints are declared on XPaths to prune set of trees. This constrains may be a value predicate, function or different types of joins. Thus, we introduce the type **Constraint** for this purpose.

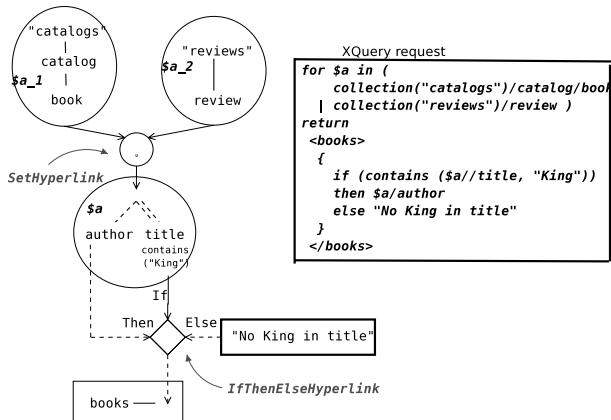
A *Constraint* is a restriction of the feasible solutions on sets of trees. It can be applied on *Nodes*, *Tree Patterns*, *Hyperlinks*, *Constraints* or *Constants*. It appears as *Predicates* or *Functions*:

- *Predicates* are constraints with a comparison operator between two element types. Linked types can be constants, nodes, tree patterns, hyperlinks or other constraints in order to compose constraints.
- *Functions* are constraints with a name and a set of links to different element types. The function name defines the type of operation to treat. Linked types can be constants, nodes, tree patterns, hyperlinks or other constraints for function composition.

Constraint representation depends of the linked element type. For a node, we put the constraint under the tag as we can see on figure 3. For a tree pattern, it is represented above it, as we saw the *count* function in figure 2 on the *AggregateTreePattern*. For hyperlinks, it depends of its type, as we will see on *JoinHyperlinks* in figure 3 a link between two nodes is annotated with a equality constraint. For constraint composition, we compose naturally at the position of the linked element (node, tree pattern and hyperlink).



**Fig. 3.** Example of node constraint



**Fig. 4.** Examples of Hyperlinks: A SetHyperLink (union) and a IfThenElseHyperlink

**Hyperlinks.** *Hyperlinks* (see figure 3 and 4) have been defined to represent additional relations between elements of the Tree Graph View:

- *Hyperlinks* link elements in *Tree Graph Views*. It represents associations by *Association Hyperlinks* or transformations by *Directional Hyperlinks*.
- *Association Hyperlinks* are *Hyperlinks* that connects two elements of the same type to represent a specific association, in order to filter results by verifying this association. There are two types of *Association Hyperlink*:
  - *Join Hyperlinks* are associations between two *Nodes* under *Constraint* pruning non relevant trees on constraints (values or order operator).
  - *Constraint Hyperlinks* are associations between *Constraints* with a Boolean connector. It forms a tree, connected to a *ReturnTreePattern* in order to keep constraints declaration level. Relevant trees must verify the connected tree of constraints, at a given declaration level.
- *Directional Hyperlinks* are injected transformations between elements. It specifies a transformation from a set of elements to a single one. There are four types of *Directional Hyperlinks*:
  - *Projection Hyperlinks* are *Node* to *Node Directional Hyperlink* representing a value projection of the given node. It can be an optional hyperlink.
  - *Specialized Hyperlinks* are *Node* to *Tree Pattern Directional Hyperlink*. It contains a mandatory or optional status. It represents the specialization of a *Node*, by specifying a new *TreePattern* which root is the given node.
  - *Generalized Hyperlinks* are *Tree Pattern* to *Node Directional Hyperlink*. It contains a mandatory or optional status. It represents a *TreePattern* generalization result set, which result is projected into the given node.
  - *Set Hyperlinks* are set of *Tree Patterns* to *Node* under *Constraint Directional Hyperlink*. It represents a set operation (Union, Intersect or Difference) between few *TreePatterns* projected on a single *Node*.
  - *IfThenElse Hyperlinks* are set of *Elements* to *Node* under *Constraint Directional Hyperlink*. Elements can be a *Node* or an *AggregateTreePattern*, and the constraint is a *Predicate* or a *Function*. It represents a conditional expression which result is deduced by the constraint status.

**Functions.** *Functions* take some parameters and give a single element in return. Into our model, we will treat only parameters with *element ()*, *boolean* and *number* types. A function is represented by a TGV, and its parameters by an *Aggregate Tree Pattern* with its function name. Variables are linked to elements by *Projection Hyperlinks*.

**Tree Graph Views.** A *Tree Graph View* (TGV) is a representation of an XQuery query containing *TreePatterns*, *Constraints* and *Hyperlinks*. Input of the TGV is given by *SourceTreePatterns*, the output is defined by the *ReturnTreePattern* (not a *AggregateTreePattern* by inheritance).

Figures 1, 2, 3 and 4 are TGV examples. For more descriptions, see [3, 8].

**Canonical XQuery to TGV.** Each queries in a canonical form can be translated to a tgv representation. All characteristics of XQuery queries correspond to an element in the tgv model. *For* clauses to STP and ITP, *where* clauses to constraints, *Constraint Hyperlinks* and *Join Hyperlinks*, *return* clauses to ATP and RTP, *let* clauses to ATP, and set and conditional operations to *Directional Hyperlinks*. All canonical XQuery queries can be translated in a TGV.

### 3 Conclusion

XQuery is an XML querying language that provides a rich expressiveness. By this way, an efficient query processing model is all the more difficult. In this paper, we describe our TGV model composed of Tree Patterns which are filters on XML documents. Thanks to this model, we are able to optimize TGV with transformation rules. Those rules rely on a mapping of *Rule Patterns* on a TGV (as a TGV on XML documents). In order to take into account physical information coming from the system, a generic annotation framework is designed on TGV. This annotation framework allows us to describe any type of information on TGVs (cost model, sources and traits localization, evaluation algorithms, etc.). The cost model is annotated on TGV in order to estimate its execution cost. It allows the optimizer to choose an optimal TGV to evaluate the query. More information can be found in [8].

The whole XQuery evaluation process is implemented in the mediator XLive [4]. All XQuery queries of the W3C use-cases [5] except the typed use-cases (STRONG) are evaluated correctly by our system, using *Tree Graph Views*.

As the TGV model is not specifically bound to a specific language (first designed for XQuery), it can be applied to any untyped queries in any language (SQL, OQL, OEM-QL, etc.) on structured or semi-structured data.

### References

1. S. Amer-Yahia, S. Cho, Laks V. S. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. In *SIGMOD*, 2001.
2. Z. Chen, HV Jagadish, L. VS Lakshmanan, and S. Paparizos. From Tree Patterns to Generalized Tree Patterns: On efficient Evaluation of XQuery. In *VLDB*, 2003.
3. T.T. Dang-Ngoc and G. Gardarin. Federating Heterogeneous Data Sources with XML. In *Proc. of IASTED IKS Conf.*, 2003.
4. T.T. Dang-Ngoc, C. Jamard, and N. Travers. XLive: An XML Light Integration Virtual Engine. In *Proc. of BDA*, 2005.
5. D. Chamberlin, P. Frankhauser, D. Florescu, M. Marchiori, and J. Robie. XML Query Use Cases, september 2005. W3C. <http://www.w3.org/TR/xquery-use-cases>.
6. HV Jagadish, LVS Lakshmanan, D. Srivastava, and K. Thompson. TAX: A Tree Algebra for XML. In *DBPL*, pages 149–164, 2001.
7. A. Sihem, C. SungRan, V. S. Laks Lakshmanan, and D. Srivastava. Tree Pattern Query Minimization. *VLDB Journal*, 11(4)::315–331, 2002.
8. N. Travers. *Optimisation Extensible dans un Médiateur de Données XML*. PhD thesis, University of Versailles, December 2006.
9. W3C. An XML Query Language (XQuery 1.0), 2005.

# Indexing Textual XML in P2P Networks Using Distributed Bloom Filters

Clement Jamard, Georges Gardarin, and Laurent Yeh

PRISM Laboratory, University of Versailles, 78000 Versailles, France  
`{clement.jamard, georges.gardarin, laurent.yeh}@prism.uvsq.fr`

**Abstract.** Nowadays P2P information systems can be considered as large scale databases where all peers can store and query data in the network. Keywords and structure indexes must be maintained. However, indexing XML documents with massive set of words brings out a major problem: The number of entries to be shipped in the network is huge. We define Distributed Bloom Filter, a data structure derived from Bloom Filters, a probabilistic data structure to test whether an element is member of a set, to summarize peer XML content and structure. Our strength is to split the traditional Bloom Filter into several segments. We rely on a DHT network to distribute these segments in a P2P network. Our measurements show that our indexing method is scalable for a large number of words, and outperforms similar methods.

**Keywords:** XML, XQuery Text, P2P Network, Database System, Bloom Filter, Indexation.

## 1 Introduction

XML and Peer-to-Peer (P2P) networks are two technologies for sharing more structured information than simple textual documents at the world scale. Among the main qualities that distinguish P2P networks, we recall dynamicity of data sources, robustness, scalability, reliability, no central administration, and no control over data placement. As XML database technology provides powerful query capabilities, and P2P networks are efficient to discover dynamically new data sources in large scale distributed mediation systems, it is valuable to couple these two technologies.

We focus on the problem of locating efficiently XML peer content on structure and value. P2P networks which were first used for simple queries as searching for filenames, must be extended to index not only text values, but also structures of XML documents. One of the main bottlenecks in P2P networks is the cost for sending in the P2P network every value to index. Thus the indexing process entails heavy network traffic.

Existing XML indexing solutions in P2P like [1] are build over solid and well-known DHT (*Distributed Hash Table*) methods ([7], [8], [6]). Most proposals take advantage of the storing primitive function `put(key, value)` for indexing one value with an associated key on the network. For indexing XML documents

in P2P networks, a natural approach is to decompose an XML document into atomic items that are indexed in the network. Then, for reducing the number of entries shipped in the network, two kind of index can be used: dense and non dense. Pathfinder [3] is representative of a dense distributed XML index. Although this approach compresses the required index size and speeds up twig queries, the number of entries to be shipped in the network remains huge.

A non dense representative index is used in DBGlobe [5], it uses Bloom Filter [2] for locating resources in a P2P network. A Bloom Filter is a bit array of size  $m$ , where bits are set to 1 for a set of  $k$  hash functions applied on values to index. Bits are checked with the functions to test membership of a value. In DBGlobe, every peer creates a Bloom Filter indexing path of its document structure. Peers groups are filtered by the union of their filters that are used to orient queries to groups that may contain relevant data. This approach is not scalable because queries must visit every Bloom Filters in the network. Moreover, only document structure is indexed.

In this paper, we propose to use Bloom Filters to index XML documents on both structure and text content. Our data model, called Distributed Bloom Filter, reduces the amount of entries indexed on the network for data localization. The filter array is split into segments to be distributed on any DHT-based network.

The remainder of paper is organized as follows: Section 2 describes our data model for designing a distributed Bloom Filter. Section 3 describes the implementation of these distributed Bloom Filter in an existing DHT-based P2P network. Section 4 presents the experimental evaluations that demonstrate the good properties of our method. Finally, Section 5 concludes the paper.

## 2 A Distributed Bloom Filter

In our system, a peer joining the overlay network publishes a description of the XML data it shares. We introduce Distributed Bloom Filters (called **DBF**) to describe peer contents. We adapt the Bloom Filter structure to *(i)* Index XML data on both structure and value criteria. *(ii)* Use the filter as a distributed index over several peers in the network.

### 2.1 Constructing a DBF from XML Documents

Our indexing method aims at solving efficiently XPath expressions with text predicates. For that, we must be able to solve structural expressions correlated with value predicates. An example of a query is `/Book//Title[. ftcontains "XML"]` looking for *title* elements descendant of a *book* with a text node containing *XML*.

Every document can be mapped to a set **VP** of *valued-paths* of type:  $/a_1/\dots/a_i[V]$ . In this *valued-path*, only the node  $a_i$  contains the value  $V$ . A *valued-path* is created for each word in a XML document text node.

For solving different kinds of XPath regular expressions, we define three kind of filters each focusing on a category of potential queries. A Bloom Filter with Path

(BFP) indexes all elements in VP for regular path expressions. A Bloom Filter Tag (BFT) is defined to solve XPath expressions containing descendant-or-self axis (e.g., `//name[. contains "Meier"]`); only  $tag[value]$  of each *valued-path* in VP are indexed. Finally, we define Bloom Filter for Words (BFW) for searching only words (e.g., `//XML`). During the publication process,  $(/a_1/\dots/a_i[V])$  is inserted in BFP,  $(a_i[V])$  is inserted in BFT, and  $V$  is inserted in BFW.

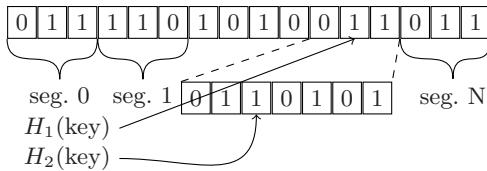
Except for the BFW, each *key* inserted in a filter is composed of a path and a value. A set of  $k$   $H_i$  function is used to determine which array entries to set to 1 in the filter for a given key. Each  $H_i$  function is the product of  $H_{pi}$ , a path coding function, and  $H_{vi}$ , a value coding function:

$$H_i(key) = H_{pi}(key.path) * H_{vi}(key.value)$$

The  $H_{vi}$  function is a typical hash function with value range from 0 to  $(S-1)$ , where  $S$  is the size of the array. For the  $H_{pi}$  function, we use a technique for encoding path inspired by Jagadish in [4]. The main idea is to map the path domain to a value between 0 and 1. More details can be found in [4]. The product result then set a bit between 0 and  $(S-1)$  to 1 in the Bloom Filter array. Thus, two paths having a different tag (e.g., `/Book/Title "XML"` and `/Article/Title "XML"`) will set to 1 different entries of the Bloom Filter.

## 2.2 Splitting a Bloom Filter for Distribution

To distribute a Bloom Filter, we split its array into segments of equal size. Segments are distributed on others peers according to their segment number. A DBF has a predefined size  $S$  shared for the whole overlay network. At the top of figure 1 we illustrate a DBF split into segments.



**Fig. 1.** Distributed Bloom Filter and Shadow Segment

Distributing segments leads to a major problem: the set of  $H_i$  could overlap randomly many segments, which would imply network traffic between peers storing the segments when testing the DBF. To avoid this, we constraint  $H_k$  functions to map on a single segment. Consequently, the first bloom function (i.e.,  $H_1(key)$ ) plays two roles, (*i*) It sets a bit to 1 for filtering purpose (i.e. the first check of the DBF), (*ii*) It determines a segment number that we use to choose the peer storing the segment. The segment number is obtained by the integer division between the size  $S$  of the filter array and the value of  $H_1(key)$ .

Once the  $H_1$  function has determined a segment number, other  $H_i$  functions maps keys (paths, valued-paths, etc.) inside this segment interval. Each segment can be viewed as a sub Bloom Filter. Therefore, when checking for a key, we first determine in which segment of the DBF the value should be, and then check other functions inside this segment. At the network point of view, it avoids contacting several peers for checking keys in segments; only one segment is needed to test the membership of a value. As Bloom Filters are prone to false positive (i.e. a key is not declared in the filter but succeeds the test), when a segment has a too high probability of false positive, the segment is replaced by a bigger segment called shadow segment. A shadow segment is illustrated at bottom of figure .

### 3 Distributing BF on a P2P Overlay

We describe in this section a P2P network overlay architecture that adapts *Chord* DHT method to implement efficiently DBFs. Most existing DHTs provide the two required primitives `put(key, value)` for indexing a value on the network, and `lookup(key)` for retrieving all values according to a given key. We can adopt any DHT methods to manage DBFs. In our architecture, a peer can play four roles. As in traditional P2P networks, a peer can be a *client*, a *server*, or a *router*. We add a fourth role: a peer can be a *controller* for managing segments of DBF.

#### 3.1 Distribution of DBF Segments

For a server peer, each created Bloom Filter (BFW, BFT, BFP) is split into segments that are distributed through the network according to the segment number. The DHT `put(key, value)` function sends the segment to a relevant controller peer using the  $H_1$  function to determine the key. As the  $H_1$  function is shared for the entire network, the peer responsible for the  $i^{th}$  segment in the network receives all  $i^{th}$  segments from every peer. It may lead to an overload of segments for a given controller peer. To avoid this bottleneck, we introduce the notion of Bloom Filter themes for publishing or querying. A user can find the relevant themes from a catalog of all existing predefined themes, shared by every peer in the network. The theme is combined with the segment number in a hash function to determine the key used for the `put(key, value)` (resp. `lookup(key)`). The message value sent through the network contains: (i) The segment of the distributed Bloom Filter. (ii) A set of Bloom Filter hashing functions ( $H_2(key) \dots H_n(key)$ ). (iii) The IP address of the sender.

The behaviour of `put(key, value)` is modified to determine the controller peer in charge of the segment and to store all information associated to the segment. The `lookup(key)` function, instead of returning all stored values corresponding to the key, is modified into a `reach(key, demand)` function that process the test on the segment and contact sources that have succeeded the test. As a Bloom Filter is prone to false positive, we must check that the server peer contains the searched value. This phase removes false positive due to the use of a Bloom Filter.

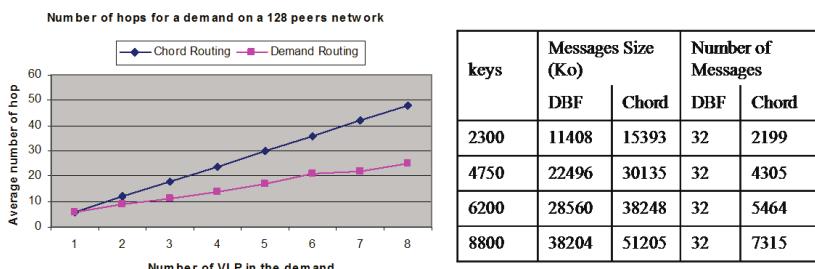
### 3.2 Query Demand Routing

Our routing process is more complex than the traditional one because the localization of a source is checked in two steps. The first step checks the DBF. As the result is prone to a false positive answer, a second step must check the source peer for an exact answer. To synchronize this process in a distributed manner we use a demand that contains all necessary query information. The demand is then resolved autonomously in the network.

A demand issued from a client peer contains a set of *value-localization-paths*. A *value-localization-path* is a searched criteria composed of a path and a value to search. Depending on the kind of path, BFP, BFT or BFW are used to resolve the *value-localization-path*. For each *value-localization-path*, the demand stores the state of the resolving process: *checkingDBF* for contacting a controller peer, or *CheckingSource* for contacting server peers. During a first phase, each controller peer responsible of a segment answering a *value-localization-path* is contacted. IP addresses of filter succeeding the test are kept in the demand. When each *value-localization-path* has been checked, server peers are contacted for a final check removing false positive and for retrieving data. Results are returned to the client peer.

## 4 Experiments

We demonstrate that our non dense index is comparable to a dense index implemented in Chord. We compared the number of messages exchanged and the density of data shipped in the network when a peer enters in the network. We use two kind of networks: a DBF network, and a classic Chord network with a basic indexing scheme (node numbering). Results are presented in the table of figure 2. As expected, the number of messages exchanged is constant (i.e, corresponding to the number of segments) and low for a network using DBF, whereas it depends on the number of keys to index in the second network. The total size of messages exchanged is lower using DBF; a single message factorizes data management (one IP address) and contains several keys whereas each message contains one key and one IP address in the second network.



**Fig. 2.** Communication messages for routing a demand and connecting a peer

The graph in figure 2 shows the number of hops needed to route a query to relevant server peers. Queries are composed of several *value-localization-path* to solve. We compared the two network configurations composed of 32 peers. We observe that a network using DBF reduces the number of hops, as a query is processed in only one demand message whereas it requires a message for every *value-localization-path* in the second network.

## 5 Conclusion

In this paper, we have proposed a new P2P indexing model based on Bloom Filters. The index is designed to locate XML sources for processing queries on both structure and value. One of our main contributions is to design a Distributed Bloom Filter, and propose techniques to split the filter for efficient and fast retrieval on a DHT-based network. We also detail how to locate relevant sources based on our Distributed Bloom Filter. Compare to other proposals, our index behaves as a non dense distributed index with word and path digests as entries. Future works are focused on methods to distribute query processing and also integrate data updates using our data model.

## References

1. S. Abiteboul, I. Manolescu, and N. Preda. Sharing Content in Structured P2P Networks. In *BDA*, 2005.
2. B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.
3. G. Gardarin, F. Dragan, and L. Yeh. P2P Semantic Mediation of Web Sources. In *ICEIS (1)*, pages 7–15, 2006.
4. H. V. Jagadish, Beng Chin Ooi, and Quang Hieu Vu. BATON: A Balanced Tree Structure for Peer-to-Peer Networks. In *VLDB*, pages 661–672, 2005.
5. G. Koloniari, Y. Petrakis, and E. Pitoura. Content-Based Overlay Networks for XML Peers Based on Multi-level Bloom Filters. In *DBISP2P*, pages 232–247, 2003.
6. S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A Scalable Content-addressable Network. In *SIGCOMM*, pages 161–172, 2001.
7. A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.
8. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

# Towards Adaptive Information Merging Using Selected XML Fragments

Ho-Lam Lau and Wilfred Ng

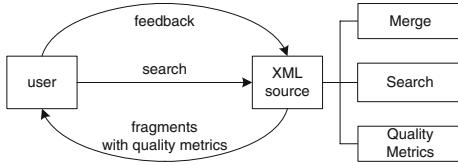
Department of Computer Science and Engineering,  
The Hong Kong University of Science and Technology, Hong Kong  
`{lauhl,wilfred}@cse.ust.hk`

**Abstract.** As XML information proliferates on the Web, searching XML information via a search engine is crucial to the experience of both casual and experienced Web users. The returned XML fragments in the list is not directly usable, if not confusing, to the users, since in most cases the XML fragments extracted from a large XML repository are incomplete, scattered and redundant. Thus, it is necessary to re-iterate the searching process based on user preferences in order to obtain more complete, detailed and usable results. In this paper, we propose a unifying framework which takes searching, merging and user preferences into account. We view search queries and fragment labeling as an input in an on-going searching process, in which the relevant XML fragments are merged into a concise form and returned to the user a ranked result list.

## 1 Introduction

As the amount and use of XML data continue to grow, searching and ranking XML data has been an important issue studied in both database and information retrieval communities [1][2][3][4][6][7][10]. Following the usual practice of handling results in Web search engines, the search results of these proposals are usually presented as a ranked list of small XML fragments to the users [1][3][11]. In practice, users do not have the schema knowledge of the underlying XML sources or have very little information of the data sources, therefore, highly structured XML queries such as XQuery FT expressions for searching are not easy for them to formulate. In addition, we recognize that the usual approach adopted by web search engines, which return a once-off list of items as the answers for a search query, is not adequate in XML setting. There are three reasons for the inadequacy. First, the target information may be scattered on the ranked list and thus it is not directly useful for the users. Second, the XML fragments can be duplicated in different ways. Third, a once-off query may not contain all desired information. In this paper, we propose a unifying framework which takes searching, merging and user preference into account.

Figure 1 shows the conceptual overview of our proposed framework. First, the user submits a query to the system and the system returns a list of fragments to the user. Then, the user selects the preferred fragments as feedback, the feedback will be merged and contribute as new search query which enlarge the



**Fig. 1.** A conceptual overview of the proposed framework

set of candidate fragments for the next iteration. Finally, the merged fragment and the new search results are returned to the user with our previously proposed notions of Quality Metrics (QMs) [9] which help users to judge the quality of fragments. We do not repeat the details in [9] here but mention that the QMs proposed are simple but effective metrics to assess the quality of individual data source or a combination of data sources, and are natural metrics to measure different dimension of the structure, data and subtrees. We contribute two main ideas related to searching XML information.

**Unifying Framework.** We propose a unifying framework that searches and merges XML fragments in a ranked result list. The search is based on a fragment, which is viewed as a set of path-key pairs.

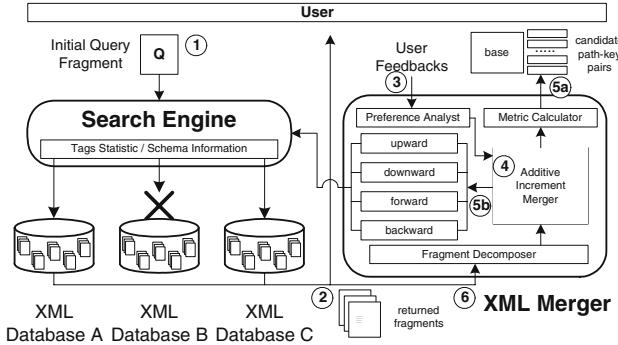
**Adaptive Merging.** We propose an adaptive merging approach and four directional searching techniques, that are able to support progressive merging the search results according to the users' continuing feedback. With the combination of searching and merging, we provide flexibility on merging that match different users' preferences.

**Paper Organization.** Section 2 presents an overview of the unified framework for searching and merging techniques. Section 3 illustrates the merging techniques and introduces the merging approach for adopting the user feedback. We conclude the framework and discuss future work in Section 4.

## 2 The Unifying Framework for Searching and Merging

In this section, we present an overview of our unifying framework for searching and merging. A *path-key pair* is an ordered pair  $(p, k)$ , where  $p$  is a path from the root to the parent node of the keyword,  $k$ . Thus, a path-key pair can be viewed as a simplified form of XPath [5]. An XML fragment is a sequence of non-repeated path-key pairs.

Figure 2 depicts the basic ideas of our framework which is able to incorporate the user preference and to support iterative searching and merging. Initially, the user submits a query to the search engine. We view the sources as the underlying XML database which collects XML fragments in a repository. Due to the space limitation, we do not describe the implementation details and the searching and ranking mechanism of the databases. However, we remark that our approach of searching and ranking techniques of XML fragments are similar to the recent work in [1,3,4,7,10].



**Fig. 2.** Overview of searching and merging XML information via key-tags

The search engine returns the list of ranked XML fragments as the raw list. The raw list is decomposed into “candidate path-key pairs” sorted by the frequency in the raw list. The top  $k$  path-key pairs is then displayed to the user (By default,  $k = 10$ ). Initially, we categorize all the path-key pairs as “unclassified”. The user feedback can be collected when he/she selects the preferred path-key pairs from the “candidate path-key pairs”, which is similar to collecting the clickthrough data in the case of HTML data [8]. However, the main difference between searching HTML data in the mentioned work and searching XML data in our approach is that an XML fragment returned can be further used as a sample for re-querying. The user feedback is collected by the “Preference Analyst” and is re-classified into two categories as follows: **preferred**, and **unclassified**.

After the (re-classification) process, the two categories of path-key pairs are passed to the AIM. The preferred path-key pairs from the user contribute the merging process in twofold. First, the AIM establishes the “result fragment” by merging the “preferred” path-key pairs. The result fragment is then returned to the user. Second, they are served as new queries (i.e. re-queries) that are sent to the four searchers of *Upward*, *Downward*, *Forward* and *Backward*. The search results of the “re-queries” will be decomposed, added into the “candidate path-key pairs” and then displayed to the user in the next iteration. More details about AIM and Directional Searching will also be given in Section 3.

### 3 The Merging and Searching Approaches

In this section, we explain our approach, *the Adaptive Increment Merging (AIM) approach*, which supports further decomposing the selected fragments from users into path-key pairs. We also discuss the four directional searchings which are able to enrich the set of candidate path key pairs in the re-querying process.

#### 3.1 The Adaptive Increment Merging Approach

The inputs of AIM are two lists: the “preferred” and “unclassified” path-key pairs and the outputs are the “result fragment”,  $R$ , and a list of reordered candidate

path-key pairs,  $C$ .  $R$  is an XML fragment resulting from merging the path-key pairs in the “preferred” list, which is possible to grow during the merging process.  $C$  is a list of path-key pairs displayed to the users for user feedback in each iteration. The path-key pairs in  $C$  are grouped according to their data sources and are sorted according to the their frequency among the list of fragments returned by the search engine. The AIM approach is shown in Algorithm 1.

---

**Algorithm 1.** Adaptive Increment Merging Approach

---

**Input:**  $R$  – the result fragment;  $P$  – a set of positive path-key pairs;  $U$  – a set of unclassified path-key pairs;  $\varrho$  – a quota variable;  $S[ ]$  – an array of sets which represent the sources;  $\varpi[ ]$  – an array of weights for the sources; // e.g.  $\varpi[j]$  is the weight of source  $S[j]$

**Output:**  $R$  – the result fragment;  $C$  – a list of candidate path-key pairs display for next iteration;

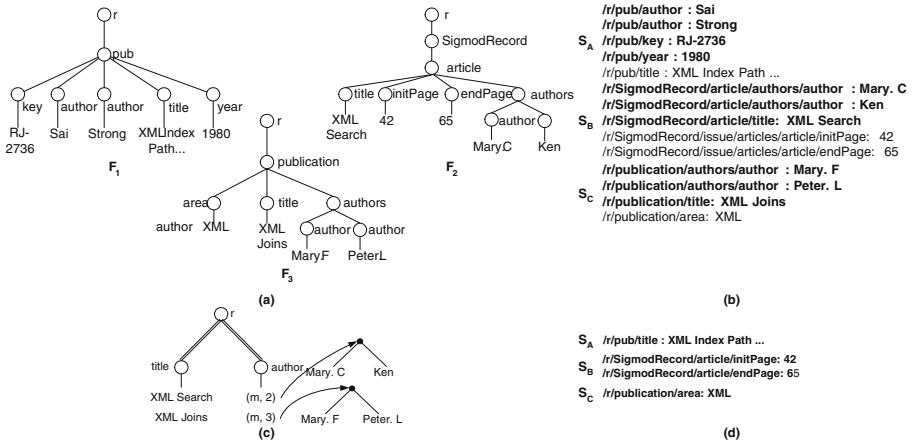
```

1 for each source $S[j]$ do
2 | $S[j] = \emptyset$;
3 | Mark $S[j]$ as negative source;
4 end
5 $C = \emptyset$;
6 for each path-key pairs $p_i \in P$ do
7 | $R = R \cup p_i$;
8 | if p_i is originated from source $S[j]$ then
9 | | $S[j] \cup p_i$;
10 | | Mark $S[j]$ as positive source;
11 | end
12 end
13 for each negative source $S[j]$ do
14 | $\varpi[j] = \varpi[j]/2$;
15 end
16 for each positive source $S[j]$ do
17 | $\varpi[j] = \frac{1 - \sum \text{weights of negative sources}}{\text{number of positive sources}}$;
18 | $C = C \cup \text{top}(\varrho \times \varpi[j]) \text{ path-key pairs in } S[j]$;
19 end
20 Perform Directional Searching using (P, U) ;
21 Return R and C ;
```

---

Consider the following example, given a query,  $Q = (\text{//author : Mary}, \text{//title : XML})$  and the fragments returned by the search engine is shown as trees in Figure 3(a).

The first step of AIM is to decompose the fragments into candidates path-key pairs and allows user to select his/her desired path-key pairs as shown in Figure 3(b). In this example, we have three sources,  $F_1$ ,  $F_2$  and  $F_3$  are from the sources,  $S_A$ ,  $S_B$  and  $S_C$  respectively. We can see that the path-key pairs are sorted according to their frequency (i.e. the number of their appearance in the raw list). For example, three path-key pairs whose path equal to “/r/pub/author” are at position 1 to 2 of the source  $S_A$ . At the first iteration, the weights for the sources are  $\{0.3333, 0.3333, 0.3333\}$ , therefore the top-3 path-key pairs from each source will be displayed to the user in next iteration. Since there are only nine path-key pairs in this case, we need an additional path-key pair in order to have ten path-key pairs for user to select, we may simply add the fourth path-key pair from either  $S_A$ ,  $S_B$  or  $S_C$ , and in this example, we add the fourth path-key pairs from  $S_A$ . The ten candidate path-key pairs for next iteration is shown in bold letters in Figure 3(b).



**Fig. 3.** (a)The returned fragments by the query  $Q$ , (b) the corresponding decomposed path-key pairs (c) merged result fragment and (d) candidate path-key pairs after the first iteration

Now, assume the user selects all path-key pairs from  $S_B$  and  $S_C$ . The result fragment is shown in Figure 3(c). We can see that the result fragment is built as expected. With the user feedback, the weight of  $S_A$  is halved and  $S_B$  and  $S_C$  share the decreased weight of  $S_A$ , the new weights are  $\{0.1667, 0.4167, 0.4167\}$ . The candidate path-key pairs are shown in bold in Figure 3(d).

### 3.2 Four Directional Searching Approaches

In this section we introduce four directional searching approaches used in the re-querying process. They are upward, downward, forward and backward searchings.

**Upward Searching.** The objective of upward searching is to find a set of fragments with similar structure but different data values. Given a query,  $Q$ , and the list of preferred path-key pairs in previous iteration,  $P$ . We formulate a re-query,  $q_i$ , for each path-key pair in  $P$ ,  $f_i = (p_i, k_i) \in P$ , where  $p_i$  is the path from the root to the parent node of the keyword,  $k_i$ . We check if  $p_i$  is located at the root of the document. If yes, we stop, since we cannot go up anymore, otherwise, the re-query is given by " $\rho_0//\rho_n : *$ ", where  $\rho_0$  is the root of  $p_i$  and  $\rho_n$  is the parent node of  $k_i$ .

**Downward Searching.** The objective of downward searching is to find a set of fragments which can provide further details according to user preference. We formulate a re-query which aims at the children or siblings of the “preferred” path-key pairs. Given a query,  $Q$ , and the list of preferred path-key pairs in previous iteration,  $P$ . We formulate a re-query,  $q_i$ , for each path-key pair in  $P$ ,  $f_i = (p_i, k_i) \in P$ , where  $p_i$  is the path from the root to the parent node of the keyword,  $k_i$ . The re-query,  $q_i$  is given by " $\rho_0//\rho_n/* : k_i$ ", where  $\rho_0$  is the root of  $p_i$  and  $\rho_n$  is the parent node of  $k_i$ .

**The Forward Searching.** The core idea of forward searching is to search relevant fragments that are ignored in the initiate query (i.e. the query submitted by the user at the very beginning) by providing more detailed query for more accurate results. Given the initiate query,  $Q$ , and the preferred path-key pairs in previous iteration,  $P$ . For each path-key pairs  $f_i = (p_i, k_i) \in P$ , if  $f_i$  does not exactly match with any path-key pairs in  $Q$ , we submit the re-query,  $r_i$ , as “ $//\rho_n : k_i$ ”, where  $\rho_n$  is the parent node of  $k_i$ .

**The Backward Searching.** The backward searching is similar to forward searching but in “opposite direction”. Backward searching aims to find information that match the initiate query,  $Q$ , but are different from the path-key pairs in  $P$ . Given a query,  $Q$ , and the list of preferred path-key pairs in previous iteration,  $P$ . For each path-key pairs  $f_i = (p_i, k_i) \in P$ , if  $f_i$  does not exactly match with any path-key pairs in  $Q$ , we submit the re-query,  $r_i$ , as “ $Q \cup //\rho_n : (\text{NOT } k_i)$ ”, where  $\rho_n$  is the parent node of  $k_i$ .

## 4 Conclusions

An interesting contribution in our proposed framework is to unify the processes of searching XML fragments and merging the users’ preferred XML fragments returned from the ranked result list. We suggest rewriting the queries using path-keys of the set of core paths in order to increase the searching coverage. We proposed the approaches of Additive Increment Merging and Directional Searching in order to generate more usable results in a progressive manner.

The ideas presented in this short paper pave the way to promote a wider use of XML data, since fragment search is simple enough for existing users to search the XML information systems. In addition, the merger provides more usable and quality information according to the users’ preferences. This paper is a ground work for many interesting issues for further study. For example, we can further examine several schemes in order to estimate path-key similarity in the merging process. This also allows us to extend our framework for searching and merging XML and HTML data, which serves as a more useful tool for searching heterogenous Web data.

## References

1. S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and content scoring for xml. In *Proc. of VLDB*, 2005.
2. J. Bremer and M. Gertz. XQuery/IR: Integrating XML document and data retrieval. In *WebDB*, 2002.
3. D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML documents via XML fragments. In *SIGIR*, pages 151–158, 2003.
4. T. T. Chinenyanga. Expressive and efficient ranked querying of XML data, 2001.
5. World Wide Web Consortium. Xquery 1.0 and xpath 2.0 full-text.
6. N. Fuhr and K. Großjohann. XIRQL: An extension of XQL for information retrieval, 2000.

7. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents, 2003.
8. T. Joachims. Optimizing search engines using clickthrough data. In *SIGKDD '02*.
9. Ho-Lam Lau and Wilfred Ng. A unifying framework for merging and evaluating XML information. In *DASFAA*, pages 81–94, 2005.
10. A. Marian, S. Amer-Yahia, N. Koudas, and D. Srivastava. Adaptive processing of top-k queries in XML. In *ICDE*, pages 162–173, 2005.
11. Martin Theobald, Ralf Schenkel, and Gerhard Weikum. An efficient and versatile query engine for topx search.

# LAPIN: Effective Sequential Pattern Mining Algorithms by Last Position Induction for Dense Databases

Zhenglu Yang, Yitong Wang, and Masaru Kitsuregawa

Institute of Industrial Science, The University of Tokyo

4-6-1 Komaba, Meguro-Ku, Tokyo 153-8305, Japan

{yangzl,ytwang,kitsure}@tk1.iis.u-tokyo.ac.jp

**Abstract.** Sequential pattern mining is very important because it is the basis of many applications. Although there has been a great deal of effort on sequential pattern mining in recent years, its performance is still far from satisfactory because of two main challenges: large search spaces and the ineffectiveness in handling dense datasets. To offer a solution to the above challenges, we have proposed a series of novel algorithms, called the LAsT Position INduction (LAPIN) sequential pattern mining, which is based on the simple idea that the last position of an item,  $\alpha$ , is the key to judging whether or not a frequent  $k$ -length sequential pattern can be extended to be a frequent  $(k+1)$ -length pattern by appending the item  $\alpha$  to it. LAPIN can largely reduce the search space during the mining process, and is very effective in mining dense datasets. Our performance study demonstrates that LAPIN outperforms PrefixSpan [4] by up to an order of magnitude on long pattern dense datasets.

## 1 Introduction

Sequential pattern mining, which extracts frequent subsequences from a sequence database, has attracted a great deal of interest during the recent surge in data mining research because it is the basis of many applications. Efficient sequential pattern mining methodologies have been studied extensively in many related problems, including the basic sequential pattern mining [1] [6] [4], constraint-based sequential pattern mining [2], maximal and closed sequential pattern mining [3].

Although there are many problems related to sequential pattern mining, we realize that the basic sequential pattern mining algorithm development is the most fundamental one because all the others can benefit from the strategies it employs, i.e. Apriori heuristic and projection-based pattern growth. Therefore we aim to develop an efficient basic sequential pattern mining algorithm in this paper.

### 1.1 Overview of Our Algorithm

For any sequence database, the last position of an item is the key used to judge whether or not the item can be appended to a given prefix ( $k$ -length) sequence.

**Example 1.** We will use the sequence database  $S$  shown in Fig. 1(a) with  $\text{min\_support} = 2$  as our running example in this paper. When scanning the database in Fig. 1(a) for

SID	Sequence	SID	Last Position of SEItem
10	a c (b c) d (a b c) a d	10	b <sub>last</sub> =5 c <sub>last</sub> =5 a <sub>last</sub> =6 d <sub>last</sub> =7
20	b (c d) a c (b d)	20	a <sub>last</sub> =3 c <sub>last</sub> =4 b <sub>last</sub> =5 d <sub>last</sub> =5
30	d (b c) (a c) (c d)	30	b <sub>last</sub> =2 a <sub>last</sub> =3 c <sub>last</sub> =4 d <sub>last</sub> =4

(a) Sequence DB

(b) Last positions of items

**Fig. 1.** Sample database

the first time, we obtain Fig. 1(b), which is a list of the last positions of the 1-length frequent sequences in ascending order. Suppose that we have a prefix frequent sequence  $\langle a \rangle$ , and its positions in Fig. 1(a) are 10:1, 20:3, 30:3, where sid:eid represents the sequence ID and the element ID. Then, we check Fig. 1(b) to obtain the first indices whose positions are larger than  $\langle a \rangle$ 's, resulting in 10:1, 20:2, 30:3, i.e., (10:b<sub>last</sub> = 5, 20:c<sub>last</sub> = 4, and 30:c<sub>last</sub> = 4), symbolized as “↓”. We start from these indices to the end of each sequence, and increment the support of each passed item, resulting in  $\langle a \rangle : 1$ ,  $\langle b \rangle : 2$ ,  $\langle c \rangle : 3$ , and  $\langle d \rangle : 3$ , from which, we can determine that  $\langle ab \rangle$ ,  $\langle ac \rangle$  and  $\langle ad \rangle$  are the frequent patterns. The *I-Step* methodology is similar to the *S-Step* methodology, which is not described here due to limited space.

Let  $\bar{D}$  be the average number of customers (i.e., sequences) in the projected DB,  $\bar{L}$  be the average sequence length in the projected DB,  $\bar{N}$  be the average total number of the distinct items in the projected DB, and  $m$  be the distinct item recurrence rate or density in the projected DB. Then  $m = \bar{L}/\bar{N}$  ( $m \geq 1$ ), and the relationship between the runtime of PrefixSpan ( $T_{ps}$ ) and the runtime of LAPIN ( $T_{lapin}$ ) in the support counting part is

$$T_{ps}/T_{lapin} = (\bar{D} \times \bar{L})/(\bar{D} \times \bar{N}) = m \quad (1)$$

Because support counting is usually the most costly step in the entire mining process, Eq.(1) illustrates the main reason why LAPIN is faster than PrefixSpan for dense datasets, whose  $m$  (density) can be very high.

## 2 LAPIN Sequential Pattern Mining

In this section, we describe the LAPIN algorithms in detail. Refer [5] for the notations and lemmas used in this paper. The pseudo code of LAPIN is shown in Fig. 2.

In step 1, by scanning the DB once, we obtain the *SE* position list table and all the 1-length frequent patterns. At the same time, we can get the *SE item-last-position list*, as shown in Fig. 1(b). In function *Gen\_Pattern*, we obtain the position list of the last item of  $\alpha$ , and then perform a binary search in the list for the (k-1)-length prefix border position (step 3). Step 4, shown in Fig. 2, is used to find the frequent *SE* (k+1)-length pattern based on the frequent k-length pattern and the 1-length candidate items. We can test each candidate item in the local candidate item list (*LCI-oriented*), which is similar to the method used in SPADE [6]. Another choice is to test the candidate item in the projected DB, just as PrefixSpan [4] does (*Suffix-oriented*).

We found that *LCI-oriented* and *Suffix-oriented* have their own advantages for different types of datasets. Thus we formed a series of algorithms categorized into two classes, LAPIN\_LCI and LAPIN\_Suffix. Please refer [5] for detail.

---

**Input:** A sequence database, and the minimum support threshold,  $\varepsilon$   
**Output:** The complete set of sequential patterns

**Function:** Gen\_Pattern( $\alpha, S, CanI_s, CanI_i$ )  
**Parameters:**  $\alpha$  = length k frequent sequential pattern;  $S$  = prefix border position set of (k-1)-length sequential pattern;  
 $CanI_s$  = candidate sequence extension item list of (k+1)-length sequential pattern;  $CanI_i$  = candidate itemset extension item list of (k+1)-length sequential pattern  
**Goal:** Generate (k+1)-length frequent sequential pattern

**Main():**

1. Scan DB once to do:
  - 1.1  $P_s \leftarrow$  Create the position list representation of the 1-length SE sequences
  - 1.2  $B_s \leftarrow$  Find the frequent 1-length SE sequences
  - 1.3  $L_s \leftarrow$  Obtain the item-last-position list of the 1-length SE sequences
2. For each frequent SE sequence  $\alpha_s$  in  $B_s$ 
  - 2.1 Call Gen\_Pattern ( $\alpha_s, 0, B_s, B_i$ )

**Function:** Gen\_Pattern( $\alpha, S, CanI_s, CanI_i$ )  
3.  $S_\alpha \leftarrow$  Find the prefix border position set of  $\alpha$  based on  $S$   
4.  $FreItem_{s,\alpha} \leftarrow$  Obtain SE item list of  $\alpha$  based on  $CanI_s$  and  $S_\alpha$   
5. For each item  $\gamma_s$  in  $FreItem_{s,\alpha}$ 

- 5.1 Combine  $\alpha$  and  $\gamma_s$  as SE, results in  $\theta$  and output
- 5.2 Call Gen\_Pattern ( $\theta, S_\alpha, FreItem_{s,\alpha}, FreItem_{i,\alpha}$ )

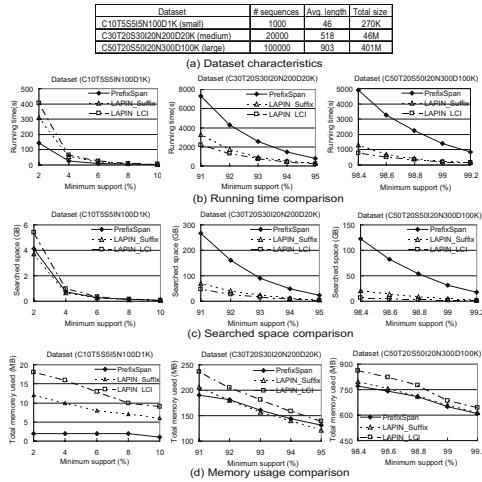
---

**Fig. 2.** LAPIN algorithm pseudo code

### 3 Performance Study

We conducted experiments on synthetic and real life datasets to compare LAPIN with PrefixSpan. We used a 1.6 GHz Intel Pentium(R)M PC with 1G memory. Refer [1] for the meaning of the different parameters used to generate the datasets. We first compared PrefixSpan and our algorithms using several small-, medium-, and large- sized datasets. The statistics of the datasets is shown in Fig. 3(a).

Fig. 3(b) and Fig. 3(c) show the running time and the searched space comparison between PrefixSpan and LAPIN and clearly illustrate that PrefixSpan is slower than LAPIN using the medium dataset and the large dataset. This is because the searched spaces of the two datasets in PrefixSpan were much larger than that in LAPIN. For the small dataset, the initial overhead needed to set up meant that LAPIN was slower than PrefixSpan. LAPIN\_Suffix is faster than LAPIN\_LCI for small datasets because the former searches smaller spaces than the latter does. However, for medium and large dense datasets, LAPIN\_LCI is faster than LAPIN\_Suffix because the situation is reversed. The memory usage of the algorithms is shown in Fig. 3(d).



**Fig. 3.** The different sizes of the datasets

**Different parameters analysis.** When  $C$  increases,  $T$  increases, and  $N$  decreases, then the performance of LAPIN improves even more relative to PrefixSpan, by up to an order of magnitude. The reason is that on keeping the other parameters constant, increasing  $C$ ,  $T$  and decreasing  $N$ , respectively, will result in an increase in the distinct item recurrence rate,  $m$ .

## 4 Conclusions

We have proposed a series of novel algorithms, LAPIN, for efficient sequential pattern mining. By thorough experiments, we have demonstrated that LAPIN outperforms PrefixSpan by up to an order of magnitude on long dense datasets.

## References

1. R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pp. 3-14, 1995.
2. M.N. Garofalakis, R. Rastogi and K. Shim. SPIRIT: Sequential PAttern Mining with Regular Expression Constraints. In *VLDB*, pp. 223-234, 1999.
3. C. Luo and S.M. Chung. Efficient Mining of Maximal Sequential Patterns Using Multiple Samples. In *SDM*, pp. 64-72, 2005.
4. J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.C. Hsu. Mining Sequential Patterns by Pattern-growth: The PrefixSpan Approach. In *TKDE*, Volume 16, Number 11, pp. 1424-1440, 2004.
5. Z. Yang, Y. Wang, and M. Kitsuregawa. LAPIN: Effective Sequential Pattern Mining Algorithms by Last Position Induction. *Technical Report*, Info. and Comm. Eng. Dept., Tokyo University, 2005. <http://www.tki.iis.u-tokyo.ac.jp/~yangzl/Document/LAPIN.pdf>
6. M. J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. In *Machine Learning*, Vol. 40, pp. 31-60, 2001.

# Spatial Clustering Based on Moving Distance in the Presence of Obstacles

Sang-Ho Park<sup>1</sup>, Ju-Hong Lee<sup>1</sup>, and Deok-Hwan Kim<sup>2</sup>

<sup>1</sup> Dept. of Computer science and Information Engineering,  
Inha University, Incheon, Korea

<sup>2</sup> Dept. of Electronic Engineering, Inha University, Incheon, Korea  
Inha University, Incheon, Korea  
parksangho@datamining.inha.ac.kr,  
{juhong, deokhwan}@inha.ac.kr

**Abstract.** The previous spatial clustering methods calculate the distance value between two spatial objects using the Euclidean distance function, which cannot reflect the grid path, and their computational complexity is high in the presence of obstacles. Therefore, in this paper, we propose a novel spatial clustering algorithm called DBSCAN-MDO. It reflects the grid path in the real world using the Manhattan distance function and reduces the number of obstacles to be considered by grouping obstacles in accordance with MBR of each cluster and filtering obstacles that do not affect the similarity between spatial objects.

## 1 Introduction

To enhance the usability of the result obtained from spatial clustering, we can extend the previous spatial clustering method in two aspects: In first aspect, we should consider obstacle constraints between spatial objects while clustering them since many obstacles exist among them. For example, building, private area, river, mountain, etc can be considered as the obstacle constraints. In second aspect, we should calculate the distance values between spatial objects using the Manhattan distance function. The road in real world can be represented as grid paths and the Manhattan distance function can reflect the moving distance of human on the grid paths.

Therefore, in this paper, we propose a new spatial clustering method, DBSCAN-MDO algorithm, based on density-based clustering. It consists of the process of grouping obstacles using MBRs of each cluster and the process of identifying obstacles to be considered using the Manhattan distance function. The process of grouping obstacles can reduce the execution time of clustering algorithm since it reduces the number of obstacles to be considered while clustering spatial objects.

## 2 Related Works

Some parts of the clustering methods have extension algorithms considering obstacles such as COD-CLARANS [4], AUTOCLUST+ [1], and DBCluC [3,5].

COD-CLARANS [4] uses the Euclidean distance function to cluster spatial objects. However, COD-CLARANS requires prior knowledge about the number of clusters in a data set and the construction of the visibility graph having the running complexity  $O(n^3)$ . AUTOCLUST+ [1] uses the Delaunay graph to model the data space. But it needs to combine different kinds of constraints. DBCluC [3,5] uses the polygon reduction method that models the obstacles as simple polygons with minimum number of line segments. But DBCluC requires the process of constructing obstruction lines. But it does not consider the time required to construct obstruction lines of obstacles in each cluster.

### 3 Clustering Based on Moving Distance

In grouping spatial objects, all obstacles between two spatial objects should be considered and the similarity values between them should be computed by moving distance of human, that is, the Manhattan distance. Human move from one data point to the other using grid paths such as cross stripes and the roads. Fig.1 illustrates a new clustering algorithm, DBSCAN\_MDO, which consists of the obstacle grouping process and the obstacle identification process.

```
Algorithm DBSCAN_MDO
Input: spatial_objects, obstacles, MinPts, Radius
1: Determine the MBRs of all spatial objects;
2: For(all obstacles)DO
3: Group the obstacles using MBRs of spatial objects;
 // GROUP_OBSTACLES algorithm.
4: ENDFOR
5: For(all pairs of spatial objects) DO
6: Identify obstacles to be considered in the presence
 of obstacle group;
7: Calculate similarity values between spatial objects
 with respect to identified obstacles;
 // CHECK_CONSIDERATION algorithm.
8: ENDFOR
9: IF((similarity value < Radius) and (count >= MinPts)) Then
10: Cluster spatial objects using the similarity values;
//DBSCAN algorithm
```

**Fig. 1.** DBSCAN\_MDO Algorithm

The first step is the computation of minimum boundary rectangles(MBRs) with respect to all spatial objects. We start by performing DBSCAN[2] using the Manhattan distance function and obtaining the MBR information(in step 1). After that, we group obstacles using the MBRs(in step 3) and identify obstacles to be considered while clustering spatial objects(in step 6). From step 7 to step 8, the algorithm

updates similarity values between spatial objects considering the identified obstacles. At the end, the algorithm returns clusters with obstacle constraints to be considered.

```

Algorithm GROUP_OBSTACLES
Input : MBRs, set of obstacles
1: min_x = MBR[i].min.x; //min x-coordinate by MBR C_i given.
2: max_y = MBR[i].max.y;
3: max_x = MBR[i].max.x;
4: min_y = MBR[i].min.y;
5: For(an obstacle ob in the set of obstacles) Do
6: Declare four variables state1,state2,state3 and state4.
 Initialize it into TRUE;
7: For(all vertices of an obstacle ob) Do
8: If(vertex.x ≤ min_x) Then state1 = TRUE & state1;
9: Else state1 = FALSE & state1;
10: If(vertex.Y ≥ max_y) Then state2 = TRUE & state2;
11: Else state1 = FALSE & state2;
12: If(vertex.X ≥ max_y) Then state3 = TRUE & state3;
13: Else state1 = FALSE & state3;
14: If(vertex.y ≤ min_y) Then state4 = TRUE & state4;
15: Else state1 = FALSE & state4;
16: ENDFOR
17: IF((state1|state2|state3|state4)== FALSE) THEN
18: Set ob as the obstacle to be considered and include
 ob into the obstacle group G_i of MBR C_i ;
19: Else Filter out obstacle ob ;
20: ENDFOR

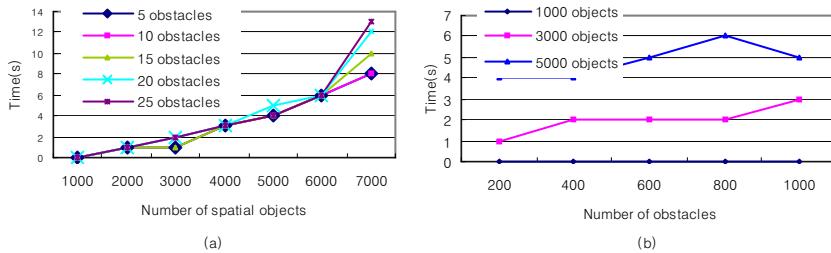
```

**Fig. 2.** GROUP\_OBSTACLES Algorithm

Fig.2 shows our grouping algorithm. Let variable  $min_x$  represent minimum value of MBR  $C_i$  in the  $x$ -axis,  $max_y$  represent maximum value of MBR  $C_i$  in the  $y$ -axis,  $max_x$  represent maximum value of MBR  $C_i$  in the  $x$ -axis,  $min_y$  represent minimum value of MBR  $C_i$  in the  $y$ -axis, respectively. The algorithm requires the computational complexity  $O(L \cdot C)$  where  $L$  is the number of obstacles and  $C$  is the number of clusters.

## 4 Experimental Evaluation and Analysis

The experiments are preformed under a Window 2000 professional on 2.40 GHz Pentium 4 CPU with main memory of 512MB and a hard disk size of 60GB. The map used for experiment have the size of  $182475.00 \times 192969.00$ . Spatial objects and obstacle dataset with complex shapes are randomly generated from GMS server. In Figs.3, the result show that our clustering method is less sensitive for the number of obstacles, because the method can reduce the number of obstacles to be considered by grouping obstacles while clustering spatial objects.



**Fig. 3.** (a) Execution time according to various numbers of spatial objects and (b) execution time according to various numbers of obstacles

## 5 Conclusion

In this paper, we address the problem of clustering spatial objects in the presence of physical constraints and propose a new extended density-based clustering algorithm DBSCAN-MDO. It has two advantages: first, it enhances the effectiveness by using the Manhattan distance function. Second, the obstacle grouping and the obstacle identification method can reduce the number of obstacles to be considered by filtering out unnecessary obstacles.

**Acknowledgement.** This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment).

## References

1. Estivill-Castro V., Lee I., "Autoclust+:automatic clustering of point-data sets in the presence of obstacles.", In International Workshop on Temporal and Spatial and Spatio-Temporal Data Mining(TSDM 2000), pages 133-146,2000.
2. Ester M., Kriegel H.-P., Sander J., Xu X., " A density-based algorithm for discovering clusters in large spatial databases with noise. In Knowledge Discovery and Data Mining, pages 226-231, 1996.
3. Osmar R. Zaiane, Chi-Hoon Lee, "Clustering Spatial Data in the Presence of Obstacles: A Density-Based Approach," *ideas*, p. 214, International Database Engineering and Applications Symposium (IDEAS'02), 2002.
4. Tung A.K.H., Hou J., Han J., "Spatial clustering in the presence of obstacles", In Proc. 2001 Int.conf. On Data Engineering(ICDE'01),2001.
5. Zaiane O.R, and Lee C.H,"Clustering Spatial Data When Facing Physical Constraints", In Proc. of the IEEE International Conf. on Data Mining, Maebashi City, Japan, pages737-740,2002.

# Tracing Data Transformations: A Preliminary Report

Gang Qian<sup>1</sup> and Yisheng Dong<sup>2</sup>

<sup>1</sup> School of Information Engineering, Nanjing University of Finance & Economics,  
Nanjing 210046, China  
[abc\\_sir@263.net](mailto:abc_sir@263.net)

<sup>2</sup> Department of Computer Science and Engineering, Southeast University,  
Nanjing 210096, China  
[ysdong@seu.edu.cn](mailto:ysdong@seu.edu.cn)

**Abstract.** We study a novel problem: tracing data transformations. That is, for a particular target data type, e.g., obtained from the output schema, we trace over the transformation specifications and extract from them the fragments that are exactly used to compute instance data of the type. Our work provides a piecemeal fashion to understand a transformation semantic, and hence would be useful for users to test, debug, and refine the transformation specifications.

## 1 Introduction

Modern information applications often need to transform data from one format to another to support cooperation, integration, and exchange of multiple information sources. On the other hand, constructing and maintaining the transformations (a.k.a. schema mappings) are labor-intensive and error-prone processes, which can involve the tasks such as testing, debugging, and refining the transformation specifications manually. This problem becomes more intractable in the XML setting. XML has been a standard format for data sharing. The XML query language, e.g., XQuery, is often used to specify the transformations of XML. Currently, many public DTDs have up to several hundreds elements and several thousand attributes. Any transformation generating XML documents for those DTDs must have a comparable complexity.

In this paper we study a novel problem: tracing data transformations. That is, for a particular target data type, e.g., obtained from the output schema, we trace over the transformation specifications and extract from them the fragments that are exactly used to compute instance data of the type. Compared with the transformations computing instance data of the whole output schema, the extracted fragments can be very simple, in terms of the given data type. So, our work provides a piecemeal fashion to understand the semantics of a complex transformation, e.g., generated by a mapping tool like *Clio* [3], and hence would be useful for the user to test, debug, and refine the transformation specifications.

We propose a mapping model, called *Macor* (*mapping* & *correlation*), through which an XML transformation is modeled as a *Macor tree* in which each node represents an *atomic transformation rule* and each edge is associated with a *correlation* (see Section 2). As a result, extraction of the transformation fragment is reduced to

matching the given data type with the Macor tree (see Section 3). In Section 4 we discuss related work. Finally, Section 5 concludes.

## 2 Mapping Model

Macor tree is an extension to a previous work presented in [4], where we introduce an incremental approach to construct schema mappings, which can be normalized into corresponding Macor trees.

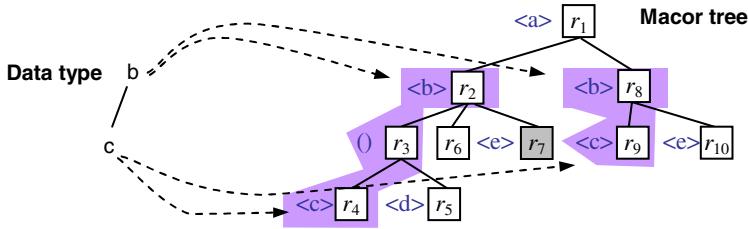
**Atomic rule** is the basic building block for the Macor tree. Using XQuery, we define it as a query returning data nodes only, i.e. *tag*, *text*, or *empty* nodes. Specifically, the atomic rule consists of only one **for**, one **return**, and one optional **where** clauses, and the **where** and **return** clauses contain no path navigations. Its main syntax is given as follows.

```
for $v1 in sp1, ..., $vn in spn (where cond)? return atomic_item
```

Here *sp* is a simple path expression with no branching predicates ([...]), *cond* is a conditional expression w.r.t. the variables defined in the rule, and *atomic\_item* refers to returned item, which can be  $<a></a>$ ,  $\$x_v$ , or  $( )$ . The symbol *a* denotes XML tags, and  $\$x_v$  is a variable bound to text values only. About *cond* we consider two kinds of equivalence comparison operators:  $=_n$  and  $=_v$  (denoted by  $\theta$ ), which compare the node identities and the values of two operands, respectively. In the following, the notation *vars(r)* refers to the variables defined in an atomic rule *r*.

**Correlation.** We organize the atomic rules into a Macor tree in which each node represents a rule and each edge is associated with a correlation, through which the atomic rules are semantically connected together: by nesting returned data nodes or by filtering data branches that do not satisfy certain conditions. We refer to the first connection as *nesting correlation* and the second as *conditional correlation*. When no confusion arises we also use a node to refer to the atomic rule it represents. Let  $r_1$  be the parent node of  $r_2$  in a Macor tree. A correlation between  $r_1$  and  $r_2$  is a pair of  $(cpath, \alpha)$ . Here *cpath* denotes a conjunction of connection path  $\$v_1 \theta \$v_2$ , where  $\$v_1 \in vars(r_1)$  and  $\$v_2 \in vars(r_2)$ . The item  $\alpha$  in the nesting correlation is null. Semantically, for each  $r_1$ 's binding tuple  $b_1$ , let  $n_1$  denote the corresponding returned data node. If there are  $i$  ( $i \geq 0$ )  $r_2$ 's bindings  $b_{2s}$  satisfying *cpath*, then the corresponding  $i$  data nodes  $n_{2s}$  returned by  $r_2$  will be nested within  $n_1$ . Conditional correlation is used to constrain the transformations, where  $\alpha$  refers to a filter. Specifically, a conditional correlation between  $r_1$  and  $r_2$  means that for each  $r_1$ 's binding  $b_1$ , if the data nodes  $n_{2s}$  returned by  $r_2$  under the connection path *cpath* satisfy the condition indicated by  $\alpha$ , then the corresponding data node  $n_1$  returned by  $r_1$  will be transformed to the target also; otherwise the node  $n_1$  together with all its branches will be filtered out.

**Macor tree.** Through the atomic rule and the correlation, we model an XML transformation as a Macor tree. Figure 1 shows such an example. Note that for the reason of brevity the correlations are omitted from the figure. As can be seen, in a Macor tree some nodes transform data, while the other nodes serve as conditions constraining the transformation. We refer to them as *d-nodes* and *c-nodes*, respectively. Each *d-node* computes a single type of data nodes, according to the **return** clause in the



**Fig. 1.** An example Macor tree and the matches between a data type and the Macor tree

corresponding atomic rule, which can be  $a$ , *text*, or  $\varepsilon$  (Note that the symbol  $a$  denotes XML tags). Notice that besides serving as a  $c$ -node, a rule returning empty data nodes can also be used as a  $d$ -node. In our example, the name of the data type has been given in Figure 1 for each  $d$ -node in the Macor tree. Through the rule  $r_3$ , the Macor tree states that the returned  $< c >$  nodes and  $< d >$  nodes are always paired.

### 3 Tracing Data Transformations

When a transformation is modeled as a Macor tree  $M$ , the exact transformation fragments for a given date type  $T$  can be extracted in terms of the matches between  $T$  and  $M$ . Each match produces a transformation for the data type  $T$ , denoted by  $M_T$ . You can regard a data type as a fragment of the output schema. In our example above, there are two matches that have been outlined using shadow in Figure 1. Let  $dn$  be a  $d$ -node of a Macor tree. The notation  $dn.cons$  refers to the conditional constraints over  $dn$ , i.e., the  $c$ -nodes nested within  $dn$ . Let  $m$  be a match of the data type  $T$  on  $M$ ,  $dn_1$  and  $dn_k$  be the roots of  $M$  and  $m$ , respectively. The transformation  $M_T$  consists of  $m$  and all  $c$ -nodes of the match. Further, to reflect the transformation context,  $dn_k$  needs to be rewritten into  $dn'_k$  and  $dn_k.cons$  needs to be rewritten into  $dn'_k.cons'$ . For a node  $dn_i$  ( $i=1..k$ ) in the path from  $dn_1$  to  $dn_k$ , let  $dn_i$  represent an atomic rule as follows.

```
for V_i in SP_i where $cond_i$ return atomic_item $_i$
```

For brevity, we write a single clause “ $V$  in  $SP$ ” instead of “ $$v_1$  in  $sp_1$ , ...,  $$v_n$  in  $sp_n$ ”. Further, let  $cpath_i$  be a connection path between  $dn_i$  and  $dn_{i+1}$ . Then  $dn_k$  is rewritten into the following atomic rule.

```
for V_1 in SP_1 , ..., V_k in SP_k
 where $cond_1$ and ... and $cond_k$ and $cpath_1$ and ... and $cpath_{k-1}$
 return atomic_item $_k$, and
 $dn'_k.cons' = dn_1.cons$ and ... and $dn_k.cons$.
```

### 4 Related Work

Some tools have recently been developed to assist the user to construct and maintain the transformations semi-automatically, e.g., by discovering candidate mappings [3], preserving their semantics as schema evolves [6], or debugging the routes of data

transformation [2]. These works are done based on logical transformation formalisms. In contrast, our work provides a way to extract physical transformation specifications, and then to facilitate the designer to understand complex transformation semantics. Data lineage tracing [1] is another important problem in modern information systems. Most of the work on this field concentrated on instance-level tracing by developing methods to generate the right queries on the source schema for a particular data value in the view, or by building annotation systems and designing some query languages for propagating annotations as data is transformed. Our work is similar to a recent work in [5], where the tracing problem is studied at meta-data level. However, their work focused on annotating the transformations and developing a language to query the transformations for some given data types. In contrast, our work concentrates on tracing and extracting the fine-grained transformations for any given data type. The work in [5] provides convenient for analyzing instance data, while our work facilitates understanding and debugging transformations. From this perspective, our work combines both the researches on schema mapping and on data lineage tracing.

## 5 Conclusion

In this work we proposed a novel problem: tracing data transformations. We designed the Macor model to represent data transformations between nested XML schemas. With Macor, a complex mapping can be modeled as a number of simple atomic rules, which are organized into a Macor tree through correlations. Any fragment of a Macor tree represents an independent transformation. Given a data type in terms of the output schema, the extraction of the exact transformations was done in terms of the matches between the data type and the corresponding Macor tree. Our work is useful for a user to understand complex transformation semantics.

## References

1. P. Buneman, S. Khanna, and W. Tan. Why and Where: A Characterization of Data Provenance. In *ICDT*, pages 316–330, 2001.
2. L. Chiticariu, and W. Tan. Debugging Schema Mappings with Routes. In *Proc. of VLDB*, 2006.
3. L. Popa, Y. Velegrakis, R. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *Proc. of VLDB*, 2002.
4. G. Qian and Y. Dong. Constructing Maintainable Semantic Mappings in XQuery. In *WebDB'05*, pages 121-126, 2005.
5. Y. Velegrakis, R. J. Miller, and J. Mylopoulos. Representing and Querying Data Transformations. In *proc. of ICDE*, 2005.
6. C. Yu and L. Popa. Semantic Adaptation of Schema Mappings when Schemas Evolve. In *Proc. of VLDB*, 2005

# QuickCN: A Combined Approach for Efficient Keyword Search over Databases

Jun Zhang<sup>1,2,3</sup>, Zhaohui Peng<sup>1,2</sup>, and Shan Wang<sup>1,2</sup>

<sup>1</sup> School of Information, Renmin University of China, Beijing 100872, P.R. China  
`{zhangjun11, pengch, swang}@ruc.edu.cn`

<sup>2</sup> Key Laboratory of Data Engineering and Knowledge Engineering(Renmin University of China), MOE, Beijing 100872, P.R. China

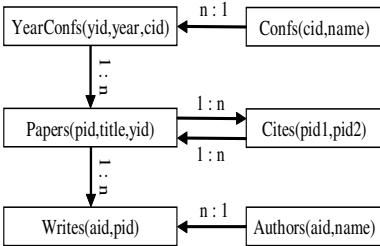
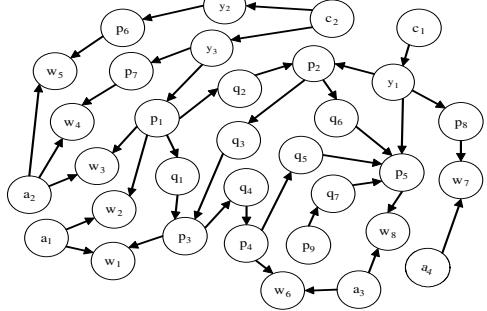
<sup>3</sup> Computer Science and Technology College, Dalian Maritime University, Dalian 116026, P.R. China

**Abstract.** Much research has been done on Keyword Search Over Relational Databases(KSORD) in recent years, and several prototypes have been developed. However, the performance of KSORD systems still is a key issue. In this paper, we propose a combined approach QuickCN for efficient KSORD. Firstly, schema graph is employed to generate Candidate Networks(CNs). Then, data graph is exploited to quickly execute CNs instead of submitting them to RDBMS. In this way, QuickCN performs more efficiently than schema-graph-based KSORD systems and consumes less memory than that by data-graph-based KSORD systems. Our experiments show that QuickCN is efficient and effective.

## 1 Introduction

In recent years, much research has been done on Keyword Search Over Relational Databases(KSORD)[1]. Many approaches have been proposed to implement KSORD techniques, and several prototypes have been developed, such as schema-graph-based Online KSORD(SO-KSORD) systems[2][3] and data-graph-based online KSORD(DO-KSORD) systems[4]. However, the performance of KSORD systems still is a key issue. On one hand, SO-KSORD systems are inefficient due to the inefficiency of JOIN operations in RDBMS because the converted SQL queries usually contain many JOIN operators. On the other hand, DO-KSORD systems consume much main memory to execute a keyword query besides the memory occupied by the data graph itself.

Therefore, we propose a combined method QuickCN(Quickly executing Candidate Network) to support efficient KSORD. Firstly, QuickCN uses database schema graph( $G_s$ ) to generate Candidate Networks(CNs)[2] which are join expressions and will be used to produce potential answers to a keyword query. CNs can also be viewed as query patterns and result patterns. Then, QuickCN employs data graph( $G_d$ ) to execute CNs instead of submitting CNs to RDBMS.  $G_d$  is a model for relational databases in which each tuple in the database is modeled as a node and each foreign-key link as a directed edge between the corresponding nodes[4]. Actually,  $G_d$  is a huge tuple-joined network generated

**Fig. 1.** DBLP Schema Graph**Fig. 2.** DBLP Data Graph Sketch Map

in advance. Intuitively, the results of any complex equi-join(a join of the form  $R \bowtie_{R.a=S.b} S$ ) expressions between primary keys and foreign keys can be found through  $G_d$ . In this way, we aim to improve the performance of SO-KSORD method and reduce the main memory required by DO-KSORD method.

Unlike SO-KSORD method, QuickCN executes CNs on  $G_d$  rather than submitting CNs to RDBMS so that it can perform more efficiently. QuickCN is also different from DO-KSORD method, such as BANKS[4]. BANKS searches the  $G_d$  without knowing of result patterns. As a result, lots of intermediate results will be produced during the search process. However, in QuickCN, the  $G_d$  search process is schema-driven as the result patterns of CNs. The adjacent nodes of each node in the  $G_d$  can be classified by their relation names and foreign-key relationship types, and the foreign-key nodes have n:1 maps with their primary-key adjacent nodes. The above two points can be exploited to reduce the search space in  $G_d$ . So, QuickCN can consume less memory than that by DO-KSORD method. Our experiments show that QuickCN is efficient and effective.

## 2 Our Approach QuickCN

QuickCN is divided into two stages. The first stage is to generate CNs. When a user keyword query  $Q_k$  comes, QuickCN generates CNs for  $Q_k$  through a breadth-first traversal of tuple set graph( $G_{ts}$ ) which extends  $G_s$  by adding Tuple Sets(TSs) created for  $Q_k$ , just as IR-Style[2] does.

**Example 1.** Take DBLP as an example. Fig. 1 shows  $G_s$  with 6 relations: C, Y, P, A, W, and Q(abbrev. for Conf, YearConf, Papers, Authors, Writes and Cites respectively). So, there are at most 4 TSs created for  $Q_k$ :  $C_{ts}$ ,  $Y_{ts}$ ,  $P_{ts}$  and  $A_{ts}$  which indicate the TSs created from the relations: C, Y, P and A respectively. The labels on the edges denote the foreign-key mapping types. Assume that part of CNs generated for  $Q_k$  are as follows:  $CN_1(Y_{ts} \xrightarrow{1:n} P \xrightarrow{1:n} Q \xrightarrow{n:1} P_{ts})$ ,  $CN_2(P_{ts} \xleftarrow{n:1} Y \xleftarrow{n:1} C \xrightarrow{1:n} Y \xrightarrow{1:n} P_{ts})$ ,  $CN_3(P_{ts} \xleftarrow{n:1} Y \xleftarrow{n:1} C_{ts}) \xrightarrow{1:n} P \xleftarrow{1:n} Q \xleftarrow{n:1} P_{ts})$ ,  $CN_4(P_{ts} \xleftarrow{1:n} Q \xrightarrow{n:1} P \xleftarrow{n:1} Y \xleftarrow{n:1} C_{ts}) \xrightarrow{1:n} W \xleftarrow{n:1} A_{ts})$ .

From Example 1, CNs can be classified into two types, one is path-shaped CN(e.g.  $CN_1, CN_2$ ), and the other is tree-shaped CN(e.g.  $CN_3, CN_4$ ).

The second stage is to execute CNs on  $G_d$ . QuickCN adequately exploits the characteristics of CNs and  $G_d$  to execute CNs. Furthermore, this stage is divided into two steps, one is to generate CN Execution Plan(CNEP), and the other is to execute CNs. Due to space limitation, all algorithms are omitted.

## 2.1 Generate CN Execution Plan

The static CNEP is generated before CN execution, and fits any keyword query. First of all, each CN is converted into a path query pattern with or without constraint queries due to its shape. For a path-shaped CN, its static CN execution plan is to construct a path query pattern and can be uniquely generated, whereas the plan for a tree-shaped CN is a path query pattern with constraint queries and can have several choices. A constraint query still is a short path query with one or more than one query nodes. When constructing execution plan for a tree-shaped CN, the following rules are applied: (a) Find the longest path as a path query pattern. Here each CN is viewed as a simple un-directed graph. (b) If there is a foreign-key-to-primary-key relationship, for example,  $a \leftarrow b(n:1 \text{ map})$ , and  $a$  is a leaf node,  $a$  is given priority to be a path query node. (c) Otherwise, if  $b$  is a leaf node,  $b$  is given priority to be a constraint query node attached to the corresponding path query node.

Then, the CNEP for each CN determines how to match the path query pattern on  $G_d$ . The following rules should also be followed: (a) The continuous  $n:1$  map edges from left to right in the path have priority to be matched. (b) The continuous  $1:n$  map edges from right to left in the path have priority to be matched. (c) Among the rest edges, if the number of  $n:1$  map edges is more than that of  $1:n$  map edges, the path query pattern will be matched from left to right, otherwise, from right to left. (d) While matching a non-leaf node with a constraint query, its constraint query should be matched at first.

## 2.2 Execute CN on Data Graph

A CN will be executed on  $G_d$  according to its static CNEP. The CN Execution is schema-driven and stack-based. A stack is created for each node in a CN plan, and a CN is matched at most the same steps as the size of CN on  $G_d$ . Suppose Fig. 2 is a portion of DBLP data graph, let's take some CNs from Example 1 as examples to demonstrate the CN execution process in detail.

For  $CN_1$  in Example 1, suppose node  $y_3$  and  $p_3$  are bound to the leaf nodes  $Y_{ts}$  and  $P_{ts}$  in the CN respectively.  $CN_1$  should be matched from right to left by its CNEP. For  $p_3$  is bound to  $P_{ts}$ , only the adjacent foreign-key nodes  $q_1$  and  $q_3$  are candidate matching nodes for Q due to the schema-driven rule, and both of them are pushed into the stack of Q. Then for the stack top element  $q_3$ , only  $p_2$  is the candidate matching node for P and is pushed into the stack of P. However,  $p_2$  has a unique adjacent primary-key node  $y_1$  which doesn't equal to the bound node  $y_3$  for  $Y_{ts}$ . So, the search process traces back till Q, the top element  $q_3$  is popped out, and the search process goes on with the new top element  $q_1$ . Then

$p_1$  is found to be matched with P, and  $y_3$  with  $Y_{ts}$  which equals to the bound node  $y_3$ . Till now, a result of " $y_3 \rightarrow p_1 \rightarrow q_1 \rightarrow p_3$ " is produced from all the top stack elements of  $CN_1$ 's nodes. Finally, the search process tracks back, but no more results are produced.

As for  $CN_2$  in Example 1, suppose node  $p_6$  and  $p_7$  are bound to the leaf node  $P_{ts}$  and  $P_{ts}$  from left to right in the CN respectively. According to the CN plan, the CN should be matched in bi-directions. Similar to the search process of  $CN_1$ , it is easy to find the result of " $p_6 \leftarrow y_2 \leftarrow c_2 \rightarrow y_3 \rightarrow p_7$ " by visiting only five nodes in the  $G_d$ .

Currently, QuickCN exploits the Global pipelined top-k Algorithm(GA) in IR-Style[2] to produce top-k results for a keyword query.

### 3 Experimental Evaluation

We ran our experiments using the Oracle 9i RDBMS on the Windows platform. BANKS[1],IR-Style[2] and QuickCN were implemented in Java. DBLP data set was used for our experiments(Fig.1). By our experiments, on one hand, QuickCN performs more efficiently than IR-Style with GA about ten times faster on average, on the other hand, QuickCN consumed always about 10 megabytes memory whereas BANKS consumed more than 200 megabytes memory on average as query keyword number increases.

### 4 Conclusions and Future Work

We have presented a combined approach QuickCN to support efficient KSORD. QuickCN uses database schema graph to generate CNs, then employs data graph to execute CNs instead of submitting CNs to RDBMS. Our experiments show that QuickCN is efficient and effective. In the future work, we try to compress the data graph and improve QuickCN with dynamic CN execution plans.

### Acknowledgements

This work is supported by the National Natural Science Foundation of China ( No.60473069 and 60496325 ), and China Grid(No.CNGI-04-15-7A).

### References

1. S. Wang, K. Zhang. Searching Databases with Keywords. Journal of Computer Science and Technology, Vol.20(1). 2005:55-62.
2. V. Hristidis, L. Gravano, Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. VLDB, 2003:850-861.
3. S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer:A System for keyword Search over Relational Databases. ICDE, 2002:5-16.
4. G. Bhalotia, A. Hulgeri, C. Nakhe et al. Keyword Searching and Browsing in Databases using BANKS. ICDE, 2002:431-440.

# Adaptive Join Query Processing in Data Grids: Exploring Relation Partial Replicas and Load Balancing

Donghua Yang, Jianzhong Li, and Hong Gao

School of Computer Science and Technology, Harbin Institute of Technology, China  
{yang.dh, lijzh, Honggao}@hit.edu.cn

**Abstract.** Query processing in data grids is a complex issue due to the heterogeneous, unpredictable and volatile behavior of grid resources. Considering the existence of multiple partial replicas for each relation and the volatile nature of grid environment, this paper investigates the issues and proposes an adaptive, load-balanced join processing approach. Analytical and experimental results show the effectiveness of our approach.

**Keywords:** Data Grids, Join query processing, Load balancing.

## 1 Introduction

The employment of data grids [1-2] provides the scientific community with fast, reliable and transparent access to geographically distributed data resources. It has been applied to a variety of fields, such as global climate simulation, high-energy physics and molecular biology.

The combination of distributed query processing and data grids is beneficial from both perspectives [3-4]. Although data grids offers a great deal of facilities for wide-area query processing, query processing is challenging due to the heterogeneous, unpredictable and volatile behavior of grid resources. As far as we know, there is little to date in the literature on distributed join query in data grids that takes relation partial replicas and load balancing into consideration. The contribution of this paper is to have proposed an adaptive join query processing approach to solve the problems.

## 2 Problem Statement

A user at any grid node issues a query that requires joining two relations  $R$  and  $S$  on the join attribute  $T$ .  $R$  and  $S$  have been split into numerous partial replicas and these replicas are present at different grid nodes. We assume that  $m_1$  different partial replicas of  $R$  ( $PR_{R8}$ ,  $R_1 \sim R_{m1}$ ), locate at  $m_1$  grid nodes  $NR_1 \sim NR_{m1}$ ,  $m_2$  different  $PR_{S8}$ ,  $S_1 \sim S_{m2}$ , locate at  $NS_1 \sim NS_{m2}$ , and some nodes,  $EN_1 \sim EN_m$ , which have tremendous processing capability and larger network bandwidth, are selected as execution nodes ( $ENs$ ) for parallel performing join operations for each pair of  $PR_{R8}$  and  $PR_{S8}$ .

Our focus in this paper is to efficiently compute  $R \bowtie S$  in data grids exploring the relation partial replicas and achieving load balancing. In general, the join of  $R$  and  $S$  is

computed through five steps. (1) Reduce  $R_1 \sim R_{m_1}$  and  $S_1 \sim S_{m_2}$  into efficient tuple sets  $R'_1 \sim R'_{m_1}$  and  $S'_1 \sim S'_{m_2}$ . (2) Select  $n_1 (n_1 \leq m_1)$  and  $n_2 (n_2 \leq m_2)$  PRs from  $R'_1 \sim R'_{m_1}$  and  $S'_1 \sim S'_{m_2}$  respectively as operand relations of join operations, satisfying  $R = R'_1 \cup R'_2 \cup \dots \cup R'_{n_1}$  and  $S = S'_1 \cup S'_2 \cup \dots \cup S'_{n_2}$ . (3) Select  $n_1 \times n_2$  ENs from  $m$  available ones for performing  $n_1 \times n_2$  join operations in parallel. (4) Transfer efficient tuple sets  $R'_1 \sim R'_{n_1}$  and  $S'_1 \sim S'_{n_2}$  to  $n_1 \times n_2$  selected ENs and perform join operations in parallel by merge-based join algorithm. (5) Transfer desired join results to user node in parallel.

Steps 4 and 5 are straightforward and are not considered in this paper.

### 3 Obtainment of Efficient Tuple Sets

In order to minimize transfer cost, we only need to transfer *efficient tuples* of each partial replicas to ENs for performing join operations. The algorithm, *Obtain-Effient-Tuples*, is divided into three phases.

- (1) Parallel get  $R_1[T] \sim R_{m_1}[T]$  and  $S_1[T] \sim S_{m_2}[T]$  at nodes  $NR_1 \sim NR_{m_1}$  and  $NS_1 \sim NS_{m_2}$  respectively by sort-based projection algorithm, where  $R_i[T]$  and  $S_j[T]$  are projections of  $R_i$  and  $S_j$  on the join attribute  $T$ .
- (2) Compute  $R[T]$  and  $S[T]$ , i.e.  $R[T] = R_1[T] \cup R_2[T] \cup \dots \cup R_{m_1}[T]$ ,  $S[T] = S_1[T] \cup S_2[T] \cup \dots \cup S_{m_2}[T]$ .
- (3) Parallel transfer  $R[T]$  to  $NS_1 \sim NS_{m_2}$ , and compute efficient tuple sets  $S'_1 = R[T] \triangleright \triangleleft S_1$ ,  $S'_2 = R[T] \triangleright \triangleleft S_2, \dots, S'_{m_2} = R[T] \triangleright \triangleleft S_{m_2}$  at  $NS_1 \sim NS_{m_2}$  in parallel. Similarly, parallel transfer  $S[T]$  to  $NR_1 \sim NR_{m_1}$  and compute efficient tuple sets  $R'_1 = S[T] \triangleright \triangleleft R_1$ ,  $R'_2 = S[T] \triangleright \triangleleft R_2, \dots, R'_{m_1} = S[T] \triangleright \triangleleft R_{m_1}$  at  $NR_1 \sim NR_{m_1}$  in parallel.

### 4 Selection of Appropriate Partial Replicas

**Definition 1.** Given a relation  $R$ , a set  $\Phi = \{R_1, R_2, \dots, R_m\}$  and a set  $F = \{R^1, R^2, \dots, R^n\}$ , where  $R_1, R_2, \dots, R_m$  are all partial replicas of  $R$ ,  $n \leq m$  and  $F$  is a subset of  $\Phi$ .  $F$  covers  $R$  if each tuple in  $R$  belongs to one partial replica in  $F$  at least, i.e.  $R = R^1 \cup R^2 \cup \dots \cup R^n$ , and for  $R^i$  and  $R^j$  ( $R^i \in F, R^j \in F, i \neq j, 1 \leq i \leq n, 1 \leq j \leq n$ ),  $R^i \not\subset R^j$  and  $R^j \not\subset R^i$ .

To minimize the response time of the join query, we should select an appropriate partial replica cover  $\langle c_r, c_s \rangle$ . The problem is formulated as follows. Given two relations  $R$  and  $S$ , two sets  $\Phi_R = \{R_1, R_2, \dots, R_{m_1}\}$  and  $\Phi_S = \{S_1, S_2, \dots, S_{m_2}\}$ , as well as  $C_R$  and  $C_S$ , where  $C_R$  and  $C_S$  are sets consisting of all relation replica covers of  $R$  and  $S$  respectively, for each element  $\langle c_r, c_s \rangle \in C_R \times C_S$ , its minimum cost  $C(\langle c_r, c_s \rangle)$  exists, where the minimum cost is defined as the minimum response time of joining  $c_r$  and  $c_s$  and  $C: C_R \times C_S \rightarrow Q^+$ . The goal of the problem is to seek an element  $\langle rc, sc \rangle$  in  $RC \times SC$  satisfying  $C(\langle rc, sc \rangle) = \text{Min}(C(\langle c_r, c_s \rangle))$ .

Once an appropriate cover  $\langle rc, sc \rangle$  and ENs are determined, the tuples in  $rc$  and  $sc$  are parallel transferred to ENs to perform join operations. To maximize the parallelism

of the join query, the number of partial replicas in  $rc$  and  $sc$  should be maximum. The algorithm *Select-Max-Replica-Cover* [6] is to seek a relation replica cover from all covers, in which the number of partial replicas is maximum.

## 5 Duplicate Removals in Selected Partial Replicas

To avoid generating duplicate join results, we have to remove duplicate tuples in  $R'_1 \sim R'_{n_1}$  so as for any tuple  $t_R$  ( $t_R \in R'$ ), it only exists in one partial replica of  $R$ . Similarly, the duplicate tuples in  $S'_1 \sim S'_{n_1}$  are processed in similar way. This ensures each join result in  $R \bowtie S$  is generated only once at all  $ENs$ .

The main idea of duplicate removals in partial replicas of  $R$  is as follows. If  $R'_a \cap R'_b \neq \emptyset$  ( $1 \leq a \leq n_1, 1 \leq b \leq n_1, a \neq b$ ) holds, we remove the tuples from the larger replica between  $R'_a$  and  $R'_b$ , i.e. a tuple  $t$  is removed from  $R'_a$  if  $|R'_{ia}| \geq |R'_{ib}|$  and  $t \in R'_a \cap R'_b$ . Similarly, duplicate removals in partial replicas of  $S$  are processed.

## 6 Adaptive Selection and Adjustment of Execution Nodes

Since the concept of *MMEM* and the algorithm *Seek-MMEM*, as well as the cost model of performing join operations and the approach for selecting  $n_1 \times n_2$   $ENs$  have been detailed in the literature [6], we focus on discussing adaptive adjustment of selection of  $ENs$  in this paper.

Data grids is an unpredictable and volatile environment, so the loads of initial selected  $ENs$ ,  $EN_1 \sim EN_{n_1 \times n_2}$ , may vary greatly with time, i.e. available CPU power degrade and available memory space become less and so on. It is necessary to determine whether to re-select at most  $n_1 \times n_2$   $ENs$ ,  $EN'_1 \sim EN'_{n_1 \times n_2}$ , and transfer remained tuples to them to continue finishing join operations. We solve this problem by constructing a weighted complete bipartite graph  $G'$  and to seek a new *MMEM*  $M'$  in  $G'$ . If  $M'$  is same to the previous *MMEM*  $M$ , the remained tuples continue being transferred to  $EN_1 \sim EN_{n_1 \times n_2}$  for performing join operations. Otherwise, the remained tuples are to be transferred to  $EN'_1 \sim EN'_{n_1 \times n_2}$ . This ensures the join operations are always parallel performed on most efficient  $ENs$  and the time cost is minimized.

## 7 Experimental Results

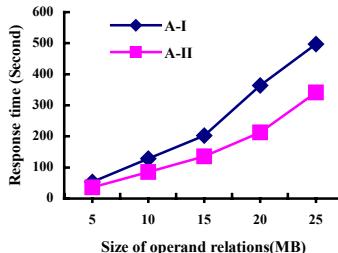
Although we get some experimental results by implementing a lot of experiments [6], we only indicate one group of experiment results due to space limitation.

Two approaches A-I and A-II are studied in this experiment to analyze the query performance related to the algorithm for adaptively adjusting the selection of  $ENs$ . In A-I,  $ENs$  are not adjusted without considering whether their loads are varied or not, i.e. once  $ENs$  are selected, the join operations are performed at them all the time. We adaptively adjust the selection of  $ENs$  according to their loads in A-II.

The process of join query is decomposed into three phases: Phase I includes getting efficient tuple sets, selecting relation replica covers and removing duplications in

reduced partial replicas; Phase II includes parallel transferring tuples to *ENs* and parallel performing join operations at selected *ENs*; transferring final join results to user node is included in Phase III.

In this experiment, we adjust the selection of *ENs* with intervals of 30 seconds. As Fig .1 shows, the response time of the query increases as the sizes of replicas become larger and the performance of A-II is well than that of A-I. Although in A-II, some time is cost for adjusting the selection of *ENs* and avoiding losing join results, the time cost in A-II is much less than that in A-I. This is because load degradation of *ENs* in A-I causes much higher time cost.



**Fig. 1.** Effect of adjusting the selection of execution nodes

## 8 Conclusion

This paper proposes a novel approach for processing join query exploring relation partial replicas and load balancing in data grids. Analytical and experimental results show that the approach has high performance. Nevertheless, there are still a number of aspects requiring further investigation to improve join query processing. For example, it is not well understood how to take relation replicas and load balancing into consideration in processing multi-join queries.

**Acknowledgements.** We would like to thank the National Natural Science Foundation of China under Grant No.60533110 and 60473075, the Program for New Century Excellent Talents in University under Grant No.NCEF-05-0333, the Key National Science Foundation of Heilongjiang Province under Grant zjg03-05 and the National Science Foundation of Heilongjiang Province under Grant F0208 for their supports.

## References

1. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure. San Francisco, CA, Morgan Kaufmann (2003)
2. Cherenvak, A., Foster, I., Kesselman, C., et al.: The Data Grid: Towards an architecture for the Distributed Management and Analysis of Large Scientific Datasets. Journal of Network and Computer Applications (2001) (23) 187-200

3. Smith, J., Watson, P., Gounaris, A., Paton, N.W., Fernandes, A.A.A., Sakellariou, R.: Distributed Query Processing on the Grid. International Journal of High Performance Computing Applications (2003) (179) (4) 353-367
4. Gounaris, A.: Resource Aware Query Processing on the Grid. Ph.D. Thesis
5. Yang, D.H., Li, J.Z., Rasool, Q.: Join Algorithm Using Multiple Replicas in Data Grid. In: Fan, W.F., Wu, Z.H., Yang, J. (eds.): Proceedings of the International Conference on Advances in Web-Age Information Management, Springer-Verlag, Berlin Heidelberg New York (2005) 416-427
6. Yang, D.H., Li, J.Z.: Adaptive join query processing in data grids: exploring relation partial replicas and load balancing. <http://db.cs.hit.edu.cn/donghua~/adaptive.pdf> (technical report)

# Efficient Semantically Equal Join on Strings

Juggapong Natwichai, Xingzhi Sun, and Maria E. Orlowska

School of Information Technology and Electrical Engineering  
The University of Queensland, Brisbane, Qld 4072, Australia  
[{jpn,sun,maria}@itee.uq.edu.au](mailto:{jpn,sun,maria}@itee.uq.edu.au)

**Abstract.** In this paper, we address a data-level data integration problem where the compared data is semantically equivalent but the data native representation of the values of given attributes is different. We assume that a semantic relationship between potentially used terms is established by a human expert prior to the designed computations, and is represented in an auxiliary table built and maintained for each attribute. We introduce a notion of *semantically equal join (SEJ)*, which is the join operation based on pre-defined semantic relationship. Our goal is to propose a solution for SEJ that can be supported by a standard SQL.

## 1 Introduction

The data integration issue is one of the most challenging problems that computer science and IT practitioners face in the last decade or so. Aside from the integration problems at schema level [1], the data-level integration problems need to be addressed. The problem at this level exists due to: potential mismatch of attributes' domains, adopted strings to represent attributes' values, as well as conventions to express the data fields. Additionally, the data mismatch may be caused by many other reasons, such as, for instance, typing errors which are addressed as the approximate string join problem [2][3].

In this paper, a data integration problem where two given relations are joined with each other on some attributes is considered. In this context, we address the *synonym mismatch problem*, i.e., the values of join attributes are semantically equivalent, but unable to match due to different representation, e.g. different abbreviation standards. Naturally, on such the attribute domain, the *semantic equivalence relationship* between strings needs to be pre-defined. We call the join operation that are based on this semantic equivalence relationship as *semantically equal join (SEJ)*.

For example, from Figure 1a,  $R_1$  and  $R_2$  are two relations which contain the medical records of two different hospitals respectively. Assume that one issues a query to find pairs of patients who have the same type of disease. Due to the fact that in medical domain, strings “Tumours”, “Kunb”, “Neoplasms” are semantically equivalent and so are the strings “Mad Cow” and “B.S.E.” (Bovine Spongiform Encephalopathy), the result of the query should be the relation given in Figure 1b. We can see that this query result can be obtained by joining two

PID	Desease	PID	Desease	PID	Desease
P <sub>1</sub>	Tumours	Q <sub>1</sub>	Neoplasms	Q <sub>1</sub>	Neoplasms
P <sub>2</sub>	Kunb	Q <sub>2</sub>	B. S. E.	Q <sub>2</sub>	B. S. E.
P <sub>3</sub>	Mad Cow	Q <sub>3</sub>	Bird Flu	Q <sub>3</sub>	Bird Flu
P <sub>4</sub>	Bird Flu				

**R<sub>1</sub>**                   **R<sub>2</sub>**

**a) Example of relation**                   **b) SEJ result**

**Fig. 1.** SEJ Example

relations  $R_1$  and  $R_2$  on their common attribute “Disease” such that the join condition is true iff the values of the join attribute are semantically equivalent.

In the rest of this paper, the focus is to formally define the SEJ operation and to propose an efficient approach to compute SEJ in a standard SQL environment which is shown in Section 2. Finally, we provide concluding remarks in Section 3.

## 2 Semantically Equal Join Approach

Let us first consider two relations  $R_1(A_1, \dots, A_m)$  and  $R_2(B_1, \dots, B_n)$  such that attributes  $A_1$  and  $B_1$  have the same domain  $Dom$  which is a set of strings.

**Definition 1.** *Semantic equivalence*, denoted as  $\approx$ , is a relation defined on a given domain  $Dom$  such that for every  $a, b \in Dom$ ,  $a \approx b$  iff  $a$  is a synonym of  $b$ . Also, we define  $a$  and  $b$  are strict semantically equivalent, denoted as  $a \sim b$ , if  $a \approx b$  and  $a \neq b$ .

*Example 1.* Consider the values of the join attribute (Disease) in Figure 1.  $Dom$  in this example is the string set {Tumours, Kunb, Neoplasms, Mad Cow, B.S.E., Bird Flu}. We can define the semantic equivalence relation  $\approx$  on  $Dom$  as Figure 2. Also observe that in this example, the strict semantic equivalence relationship is shown as the right-hand-side of the union operator.

$$\begin{aligned} & \{(Tumours, Tumours), (Kunb, Kunb), (Mad Cow, Mad Cow), (Bird Flu, Bird Flu), \\ & (Neoplasms, Neoplasms), (B.S.E., B.S.E.)\} \cup \\ & \{(Tumours, Kunb), (Kunb, Tumours), (Tumours, Neoplasms), (Neoplasms, Tumours) \\ & (Kunb, Neoplasms), (Neoplasms, Kunb), (Mad Cow, B.S.E.), (B.S.E., Mad Cow)\} \end{aligned}$$

**Fig. 2.** Semantic equivalence

**Definition 2.** Given a relation of semantic equivalence  $\approx$ , *semantically equal join (SEJ)*, denoted as  $\bowtie$ , is an theta join operator in the relational algebra such that the theta comparison operator is  $\approx$ .

To perform the semantically equal join between two tables  $R_1$  and  $R_2$  on attributes  $A_1$  and  $B_1$ , the semantic equivalence relation on domain  $Dom$  needs to be predefined. Since it is straightforward to observe that a string can

semantically join with itself, we define a relation  $Au$ , called *auxiliary table*, to only specify the relationship of strict semantically equivalent, i.e., to give the information of synonyms in  $Dom$ . From Definition 1, it is easy to observe that semantic equivalence,  $\sim$ , is an *equivalence relation*<sup>1</sup> which partitions  $Dom$  into a set of *equivalence classes*  $C_i$ ,  $i = 1 \dots l$ . We define the set  $\Omega$  of *non-trivial equivalence classes*, formally,  $\Omega = \{C_i \mid |C_i| > 1\}$  where  $i = 1 \dots l$  (only  $|C_i| > 1$  provides the information of synonyms). According to the above discussion,  $Au$  can be regarded as the representation of the set  $\Omega$ .

For brevity, we will only discuss the SEJ on a single attribute which can be extended to  $k$  attribute readily. The problem statement is given as follows.

**Problem statement:** Let  $R_1(A_1, \dots, A_m)$  and  $R_2(B_1, \dots, B_n)$  be two relations such that attributes  $R_1.A_i$  and  $R_2.B_i$  ( $i = 1 \dots k$ ) have the same domain  $Dom_i$ , where  $Dom_i$  is a set of strings. For each  $Dom_i$  ( $i = 1 \dots k$ ), let  $\sim^i$  be the equivalence relation defined on  $Dom_i$  and  $Au_i$  be the auxiliary table that represents the corresponding strict semantically equivalent relationship. The semantically equal join (SEJ) problem is: given relations  $R_1, R_2$  and auxiliary table  $Au_1, \dots, Au_k$ , express  $R_1 \tilde{\bowtie}_{A_1 \sim^1 B_1, \dots, A_k \sim^k B_k} R_2$  by a single SQL statement.

## 2.1 Schema of Auxiliary Table and Semantically Equal Join

We propose a structure of auxiliary tables, called the group-based approach for the schema of the auxiliary table. In this approach, given the set  $\Omega$ , the auxiliary table  $Au$  is defined as  $Au(Gid, P) = \{(i, a) \mid C_i \in \Omega \text{ and } a \in C_i\}$ . In Example 1, the auxiliary table is given in Figure 3a.

Gid	P
1	Tumours
1	Kunb
1	Neoplasms
2	Mad Cow
2	B.S.E.

a) Group-based  $Au$  example

```
SELECT *
FROM R1, R2
WHERE A1=B1 OR
((SELECT Gid FROM Au WHERE P=A1)
=(SELECT Gid FROM Au WHERE P=B1))
```

b) Sub-query-based

Fig. 3. SEJ Approach

Given the schema of  $Au$ , the RA expression for the semantically equal join for this schema is:

$$\begin{aligned} R_1 \tilde{\bowtie} R_2 \equiv & (R_1 \bowtie_{A_1=B_1} R_2) \cup \\ & ((R_1 \bowtie_{A_1=P} Au) \bowtie_{Au.Gid=Au'.Gid} (R_2 \bowtie_{B_1=P} Au')) \end{aligned} \quad (1)$$

where  $Au'$  is the copy of  $Au$ .

<sup>1</sup> As a equivalence relation, “ $\sim$ ” has the following property: 1)  $a \sim a$  for all  $a \in Dom$ , 2)  $a \sim b \rightarrow b \sim a$  for all  $a, b \in Dom$  3)  $(a \sim b) \wedge (b \sim c) \rightarrow a \sim c$  for all  $a, b, c \in Dom$ .

## 2.2 Query for Implementing SEJ

Given the relational algebra operation in Equation ⑩, there are multiple ways to build an SQL statement to compute the semantically equal join. Although generally, different queries may have the same execution plan in a given DBMS, in many cases, the way how to write query will affect the execution plan and the query performance.

We propose an SQL statement which can efficiently compute semantically equal join in Figure 3b, called *sub-query-based SQL statement*. The semantic meaning behind of this query is that two tuples can be joined together if either the values of the join attributes are the same, or they belong to the same group in the auxiliary table. Note that when both values of  $A_1$  and  $B_1$  are not in the auxiliary table, the condition after ‘OR’ will compare two Null values. In the standard SQL, the result of Null=Null is ‘unknown’, which will not make two tuples join.

For the I/O cost for computing the SEJ, when  $R_1$  and  $R_2$  are joined, for any pair of tuples which can not equally join, auxiliary table is accessed by the primary index (on the attribute  $P$ ) twice to check the semantic join condition. Note that in practice, auxiliary table is not very large and often can be caught in the buffer. In this case, the I/O cost of sub-query-based SQL is close to the cost of equal join of  $R_1$  and  $R_2$ . For above discussion, we can see the sub-query based approach requires less disk I/O when the size of  $Au$  is small.

## 3 Conclusion

In this paper, we have addressed the problem of synonym mismatch in the context of data integration, where the data from different sources are semantically equivalent but the data native representation is different. The originality and contributions of our work include the following aspects: 1) We have introduced the semantic equivalent relationship to resolve the synonym mismatch problem. In fact, this study can also complement previous works of approximate string join that mainly focuses on resolving typo mismatch. 2) We have formally defined the concept of SEJ and proposed an efficient approach to implement the SEJ operator in the standard RDBMS by a single SQL statement.

## References

1. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. *The VLDB Journal* **10** (2001) 334–350
2. Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. In: Proceedings of 27th International Conference on Very Large Data Bases, Morgan Kaufmann (2001) 491–500
3. Gravano, L., Ipeirotis, P.G., Koudas, N., Srivastava, D.: Text joins for data cleansing and integration in an rdbms. In: Proceedings of the 19th International Conference on Data Engineering, IEEE Computer Society (2003) 729–731

# Integrating Similarity Retrieval and Skyline Exploration Via Relevance Feedback

Yiming Ma and Sharad Mehrotra\*

Dept. of Information and Computer Science, University of California,  
Irvine, CA, USA

**Abstract.** Similarity retrieval have been widely used in many practical search applications. A similarity query model can be viewed as a logical combination of a set of similarity predicates. A user can initialize a query model, but model parameters or the model itself may be inadequately specified. As a result, a retrieval system cannot guarantee that it has presented all the relevant tuples to the user. In this paper, we propose a framework that integrates the similarity retrieval and skyline exploration. Using the relevance feedback as a way to constrain the search space, our framework can intelligently explore only a necessary portion of data that contains all the relevant tuples. Our framework is also flexible enough to incorporate model refinement techniques to retrieving relevant results as early as possible.

## 1 Introduction

Similarity retrieval is attractive since it presents results quickly to the user in relevance order and allows the search to stop when enough results are seen (as contrasted to a potentially large collection of results from which a user must choose the relevant ones). A fundamental weakness of the similarity query model is that it requires a user to accurately specify the model parameters which, given the complexity of search spaces, might be a difficult (or impossible) task. If the user does not specify the parameters accurately, the system cannot guarantee to retrieve all the relevant results. For instance, if the user stops the search after retrieving  $k$  objects because the latest objects retrieved were irrelevant, there is no guarantee that the unseen objects are also irrelevant. It is possible that the *best* answer resides in the unseen results. One approach that can guarantee the answer set containing best results is by using a skyline [1]. In a skyline setting, the system pessimistically assumes no knowledge of the query model; it knows only the similarity predicates in a user's search. Instead of returning objects based on relevance to the user, a skyline operator returns a set of objects that are not dominated by any other object in at least one similarity dimension (formed by a similarity predicate). This way, the top result is guaranteed to be in the return set (irrespective of the user's similarity query model). While skyline

---

\* This research was sponsored by NSF Award number 0331707 and 0331690 to the RESCUE project.

retrieval offers the guarantee to the best results, it suffers from three problems: (1) the size of the return set (the skyline) may be large and the skyline size increases as the dimensionality of the similarity space increases, (2) since the returned objects are no longer based on any ranking criteria (within a skyline), if a user stops the search prior to viewing the entire skyline, the top results may be missed, and (3) it is not an interactive process (i.e., does not consider relevance feedback).

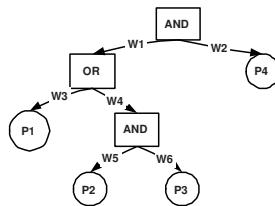
In this paper, we build a search strategy that combines the positive aspects of both similarity retrieval and skyline retrieval into one single technique, so that we can retrieve results in the order of relevance, yet support the notion of completeness. The key aspect of our technique relies on exploiting the relevance feedback gathered from a user. We use relevance feedback in two ways, which also represent the major contributions of this work:

- Initialized by a ranked retrieval, we use irrelevant (negative) feedback to progressively form an interactive skyline (I-Skyline), which dynamically constrains the search space (Section 3).
- Using both relevant (positive) and irrelevant (negative) feedback, we introduce query model refinement techniques to improve the search quality; so that the relevant tuples will be retrieved more effectively from the search space bounded by I-Skyline (Described in the full version 3).

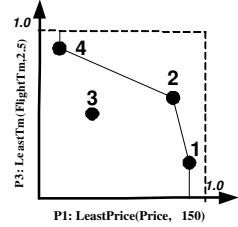
## 2 Related Work

To the best of our knowledge, no previous work uses relevance feedback as a bridge to effectively integrate similarity retrieval and skyline retrieval. Our work significantly differs from these retrieval and refinement approaches (e.g., [7,4]) since we focus on the existing query formulation, and attempts to give a user a sense of the query completion as we dynamically prune the search space based on the user feedback. In addition, our refinement techniques are built on top of the dynamic search space. The techniques are not available in any of the refinement systems since those systems may alter the query formulation (e.g., predicate addition/deletion). During a search session, a user may never know if his initial query has been completed or not. In fact, he may be even confused by the returned tuples since he does not know the exact query formulation used to rank the returned tuples. Our work is also very different from the work on skyline (e.g., [5]), which mainly focus on the efficiency issues. Many of these techniques assumes the ranking function is defined in a feature/data space, and an index structure (e.g., R-tree) is available in the feature space. By making these assumptions, it imposes limitations on the query, such that every attribute can appear at most once in the query, and similarity predicates can only use distance measures in the feature space. Therefore, it does not support general similarity queries. Furthermore, it does not exploit the relevance feedback information to bound the search space or refine the ranking function.

P1: MinPrice (Price, 150)  
P2: MinStop (#Stop, 0)  
P3: MinFlightTm(FlightTm, 2.5)  
P4: ArrTm (ArrTm, 1500)  
Query Model:  $(P1 \vee (P2 \wedge P3)) \wedge P4$



**Fig. 1.** Example Similarity Predicates and Query Model



**Fig. 2.** A Logic Tree  
 $(P1 \vee (P2 \wedge P3)) \wedge P4$

**Fig. 3.** Skyline on P1,  
P3

Terminology	Explanations
Similarity Predicate	Similarity based logic predicate.
Query Model ( $Q$ )	A logic combination function built on top of the similarity predicates.
Logic Combination Function	Used exchangeably with query model.
Monotone Function	Defined in [2], a query model is also a monotone combination function.
Parameters in Query Model	Weights and $p$ values used in the query model (P-Norm [6]).
Logic Tree ( $LT$ )	An operator tree representation of query model.
Full Similarity Space ( $FS$ )	A similarity space defined by all the similarity predicates.

**Fig. 4.** Terminologies

### 3 I-Skyline Framework

In Figure 4, we summarize the concepts and the terminologies used in this paper. In this paper, we assume that users can specify all the similarity predicates of interest to their information needs. We focus on the similarity query model which, when similarity semantics are involved, can be difficult to specify correctly. For instance, a flight ticketing database has four attributes: price, number of stops, flight time, and time of arrival. A typical query is to find flights that conform to a certain desirable hypothesis expressed as a similarity query. In our example, the search has four similarity predicates: *MinPrice*, *MinStop*, *MinFlightTm* and *ArrTm* with obvious semantics. Figure 1 shows an example of similarity predicates and a query model. Given a data tuple, the query model aggregates the predicate level scores to a single relevance score using a set of logical operators (*AND*, *OR*). A user invokes this query to find a flight with the cheapest fare or least number of stops with the shortest flight time; the flight should also arrive in Seattle at around 3pm.  $(P1 \vee (P2 \wedge P3)) \wedge P4$  nicely captures this search request. In general, a query model using logical operators can be always viewed as an operator tree. Figure 2 shows such an operator tree; an internal node is a logical operator, and a leaf node corresponds to a similarity predicate. Outgoing edges from an internal node connect the components used in a logical operator. In this paper, we use P-Norm [6] to interpret logical operators. Because of the tunable parameters (weights and P values), the P-Norm can be expressive. However, if the initial parameter settings of a query model differ from the ideal ones, the order of the returned tuples can change considerably. Skyline could be utilized to retrieve the best answer. Given any  $d$  similarity predicates, if we

---

Input: Database( $D$ ), FullSimSpace( $FS$ ), Monotone Comb. Func. ( $F$ )  
Output: I-Skyline, RelevantSet

```

1: I-Skyline = \emptyset , RelevantSet = \emptyset
2: $RL = \text{compute.RankedList}(D, FS, F)$
3: for each tuple t in RL do
4: if t is NOT Dominated by any tuple $t_2 \in$ I-Skyline then
5: $FB = \text{getFeedbackFromUser}(t)$
6: if $FB == \text{Irrelevant}$ then
7: I-Skyline.insert(t) // Grow I-Skyline set.
8: else
9: RelevantSet.insert(t) // Grow Relevant set.
```

---

**Fig. 5.** I-Skyline\_Base

define a  $d$ -dimensional space on these predicates and project data points into the space using their similarity scores, the skyline is guaranteed to consist the best point under any monotone query models [1]. Figure 3 shows an example of a 2-dimension space defined on predicates P1 and P3 in Figure 1. A skyline retrieval will return tuples 1,2 and 4.

Instead of retrieving one best record as the skyline retrieval, in this paper, the goal is to retrieving all the relevant tuples. We assume there is an optimal query formulation  $Q^{opt}$ . The relevant tuples are a list of top tuples that having similarity scores above a threshold  $\tau$ . Without knowing the  $Q^{opt}$  and  $\tau$ , the problem is how to retrieve all the relevant tuples with minimum number of irrelevant tuples given an initial query  $Q$ .

We now present the framework I-Skyline algorithm called *I-Skyline\_Base* in pseudo-code (Figure 5). We first define I-Skyline as a skyline on all irrelevant tuples in a given full similarity space  $FS$  (Figure 4). The algorithm *I-Skyline\_Base* sits in between a ranked retrieval system (line 2) and a user. It interacts with the user (line 5) and progressively selects tuples that the user needs to see (line 3 to 9). During the process, only two sets – I-Skyline set and Relevant set – are dynamically constructed (line 7 and line 9). It can be formally proved that these two sets contain necessary (optimal) set of tuples that the user needs to interact with if there is no prior knowledge to the query model.

The baseline algorithm can be easily extended and enhanced in various ways such as exploiting the partial knowledge provided to the query structure or aggressively improving the retrieval quality by incorporating various refinement and learning techniques. In the full version of this paper [3], we provide detail discussions and extensive evaluations to these strategies.

## References

1. S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*. 2001.
2. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
3. Y. Ma and S. Mehrotra. I-skyline: A systematic approach in integrating similarity retrieval and skyline exploration via relevance feedback. *UCI Technical Report available at <http://www.ics.uci.edu/~maym/publications/iskyline.pdf>*, 2006.

4. Y. Ma, S. Mehrotra, and Q. Zhong. RAF: An Activation Framework for Refining Similarity Queries Using Learning Techniques. In *DASFAA*, 2006.
5. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, 2003.
6. G. Salton, E. Fox, and H. Wu. Extended boolean information retrieval. *Communications of the ACM*, 1983.
7. L. Wu, C. Faloutsos, K. Sycara, and T. Payne. FALCON: Feedback adaptive loop for content-based retrieval. In *VLDB*, 2000.

# An Image-Semantic Ontological Framework for Large Image Databases

Xiaoyan Li, Lidan Shou, Gang Chen, and Kian-Lee Tan

College of Computer Science, Zhejiang University, Hangzhou 310027, P.R. China

School of Computing, National University of Singapore, Singapore 117543

## 1 Introduction

In the past decade, the Internet has rapidly become a most prevalent platform for information sharing and data communications all over the world. This trend has been enhanced by the wide proliferation of home-used digital photos and videos. Although the Web has seen numerous applications for photo storage, sharing, and searching, few image retrieval systems provide satisfactory search service. The so-called *semantic gap* and *curse of dimensionality* are two major barriers that existing technologies cannot adequately address. In this paper, we propose a framework to achieve effective and efficient content-base retrieval on images by addressing the above two issues. That is, we need to narrow the *semantic gap* while mitigating the *curse of dimensionality*.

In our proposed approach, we capture the descriptive properties of images in two classes of features: *textual descriptions* and *visual features*. The textual descriptions of an image include the content-related annotation words which carry the semantics of the image. The visual features contain visual properties such as colors, distribution of colors, and so on.

Our method for bridging the *semantic gap* is motivated by the following observation: In a CBIR system, the textual descriptions usually carry more semantics of images compared to the visual features. As a consequence, textual descriptions can describe high-level abstractions and concepts, while visual feature similarity measure is effective only when their semantics are well-correlated. To bridge the *semantic gap*, we need to integrate both classes of features together.

Our dimensionality reduction technique is implemented in two phases. In the first phase, we use an ontological structure (as a simple example shown in Fig. 1) to capture the meaning of the textual data of images. This structure hierarchically organizes the concepts and their interrelationships for images. Searching in this structure is based on keywords. In the second phase, visual similarity computation is performed at a much more limited scale. That is, we perform visual comparison among only a small subset of images called an *Atomic Semantic Domain (ASD)*, which share the same *semantic unit*. The visual descriptors for different images of the same semantic unit are more selective and therefore the intrinsic dimensionality in this subset of images becomes smaller.

The remainder of this paper is organized as follows: In section 2, we present our image-semantic ontological framework. We describe the construction process

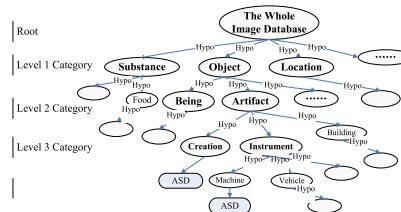
of the framework in section 3. The retrieval technique is proposed in section 4. Finally, we present the concluding remarks in section 5.

## 2 The Image-Semantic Ontological Framework

In this section, we introduce the three main data (feature) structures for our framework, namely a *Lexical Hierarchy* for organizing words, an *Image-Semantic Ontology* for organizing high-level semantics of images, and a set of ASDs for organizing the visual features within each semantic unit.

**The lexical hierarchy for annotation words:** The binary lexical relations captured in the *Lexical Hierarchy* may have three types: *synonym*, *hyponym* and *hypernym*. The synonymous words are grouped into one *synset*, and pointers are used to describe the relations between a synset and another (hyponym or hypernym). A word may appear in more than one synset in the Lexical Hierarchy. We mainly use nouns and some verbs to produce the hierarchy, as we intend to classify image set according to the taxonomies of semantic categories and the interrelations among these categories.

**The image-semantic ontology:** We use an ontology to organize the semantic concepts of the image collection. The annotation words are used as the data source for the ontology. By referring to the lexical hierarchy and combining the generic ontological knowledge and more domain specific semantics, the semantic ontology is able to capture the underlying ‘general knowledge’ that people have about physical objects and substances in the world. Fig 1 shows a simple sketch of our image-semantic ontology, where each node of the ontology tree represents a ‘category’. The category is more generic at the higher levels and more specific at the lower levels of the ontology. The formal definition of the representative information (the representing keyword set and corresponding weight vector) of each node  $n$  has been presented in [2]. The keyword vector of a node in the higher level is likely to include more generic keywords as hypernyms of some specific words in its offsprings.



**Fig. 1.** A simple ontological structure

**The atomic semantic domains:** Images indexed by the same leaf node of the ontological structure form a semantic unit, referred to as an *Atomic Semantic Domain*. The image semantics are well correlated within each ASD, where the

low-level visual features are more discriminative in visual similarity comparison. By adopting proper dimensionality reduction techniques, a very small number of parameters for each image is sufficient to capture the distinguishing characteristics within this semantic unit.

### 3 The Construction Process

In this section, we describe how we collect and organize the textual and visual features for our framework to generate its data structures.

**Textual data collection:** We use the annotation words of images as the input source for the Image-Semantic Ontology construction. In our proposal, the annotation words can be collected via three methods, namely *extracting from the illustration of images*, *manually adding token words*, and *automatic image annotation*, as shown in Fig. 2.

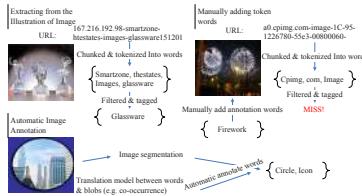


Fig. 2. Three methods for annotation words collection

**The lexical analysis and hierarchy construction:** In this work, we use the WordNet [3] as our knowledge base to perform the lexical analysis and construct the lexical hierarchy for the annotation word set. The annotation words that we collect from the image collection mainly consist of nouns, as well as some verbs and adjectives that have the potential to reduce the ambiguity when combined with nouns. The word relations discovered through lexical analysis are organized into the *Lexical Hierarchy*. Since WordNet is a full word dictionary, we need to select the main branch of the WordNet hierarchy and specify our own concepts to adapt the semantic meanings of the images.

**The image-sematic ontology construction:** The elicitation of our Image-Semantic Ontology consults the common-sense knowledge, and domain knowledge, as well as sets of complex agreement rules made by designers together with domain experts. These are the fundamental resources for constituting a potentially valuable ontology. By using the keyword set combined with its lexical hierarchy information, we apply a Generative Hierarchical Clustering (GHC) pattern [2], from generic to more specific, to construct a tree-like conceptual taxonomy (see Fig. 1) in a top-down fashion. The image dataset is subsequently partitioned from coarser to finer with respect to their annotation words.

**The atomic semantic domains:** The visual descriptor for each image is a 64-dimension normalized vector of wavelet coefficients extracted using Daubechies'

wavelets [11]. To further improve the retrieval performance, we apply a dimensionality reduction technique, the SVD (Singular Value Decomposition) method, to select the distinguishable features from the visual descriptors.

## 4 The Retrieval Algorithm

The image query is handled in two phases. The first phase of a query uses the Image-Semantic Ontology structure, to search for the nodes which contain relevant image semantics. The second phase of a query searches the ASD structures pointed to by the leaf nodes and fetches the candidate images which are visually-similar. Given a query keyword set, provided by a user or extracted from a query example image, the first phase can accelerate the image retrieval by indexing the dataset through an access method tailored to a sub-domain. An image might be indexed by multiple leaf nodes based on the results of the relevance probabilistic analysis for its annotation words. To obtain more accurate results that are both semantically-related and visually-similar, we perform the visual similarity comparison within the single relevant ASD in the second phase.

## 5 Conclusion

In this work, we propose a framework which employs an ontological structure to model and express the semantics in the image collection by using the textual features of images. The relevant image semantic unit can be quickly located by searching through this structure based on keywords. Visual similarity comparison is performed within each local semantic unit to obtain the query results. Our framework is effective in addressing the semantic gap problem in content-based image retrieval.

## Acknowledgements

This research was funded in part by the National Science Foundation of China, in grant NSFC No. 60603044.

## References

1. J. Z. Wang, G. Wiederhold, O. Firschein, S. X. Wei. Content-based image indexing and searching using Daubechies' wavelets. *Int.J.on Digital Libraries* 1(4): 311-328,1998.
2. X. Y. Li, L. D. Shou, G. Chen. A Latent Image Semantic Indexing Scheme For Image Retrieval On The Web. *WISE 2006*, LNCS 4255, pp. 315-326, 2006.
3. G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3, 235-244(1990).
4. C. Rother, V. Kolmogorov, and A. Blake. Grabcut - interactive foreground extraction using iterated graph cuts. *Proc.ACM Siggraph*, 2004.

# Flexible Selection of Wavelet Coefficients for Continuous Data Stream Reduction

Jaehoon Kim and Seog Park

Department of Computer Science, Sogang University  
1-1 Shinsu-Dong Mapo-Gu Seoul Korea 121-742  
`{chris3,spark}@dblab.sogang.ac.kr`

**Abstract.** In this article, we introduce a continuous data stream reduction method using wavelets summarization. Especially we consider storing a plenty of past data stream into stable storage (flash memory or micro HDD) rather than keeping only recent streaming data allowable in memory, because data stream mining and tracking of past data stream are often required. In the general method using wavelets, a specific amount of streaming data from a sensor is periodically compressed into fixed size and the fixed amount of compressed data (selected wavelet coefficients) is stored into stable storage. However, our method flexibly adjusts the number of selected wavelet coefficients for each local time section. Experimental results with some real world data show that our flexible approach has lower estimation error than the general fixed approach.

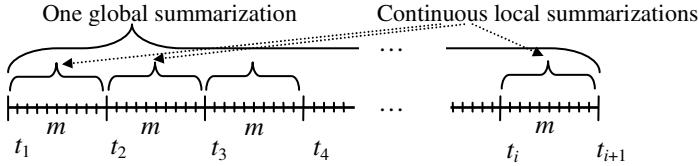
## 1 Introduction

Recently a great deal of attention has been driven toward processing data stream in mobile computing, ubiquitous computing, and sensor network. For example, mobile healthcare is to use mobile device equipped with biosensors and advanced wireless communication technology (3G/4G) to analyze the chronic conditions of certain disease and detect health emergencies [5]. A traffic control system with smart sensors (or called motes [1]) at major crossroads enable us to monitor and analyze traffic data in real time.

The infinite extent of streaming data from sensors makes it necessary to periodically store the past data stream in stable storage, and queries on this past data are also important. These queries include database queries over the past data stream, together with analysis and data mining. However, it is impractical to store all the data because stable storage still has restricted and low capacity in mobile device and motes. Therefore, the data reduction method such as wavelets [4, 6], histograms, and sampling can be considered to store much more data.

*Problem Definition.* Since streaming data are continuous and endless, the limits of data to be summarized are ambiguous. Therefore, we must calculate local summaries (i.e., the  $m$  wavelet coefficients) periodically for fixed amounts of data, and then store them independently into stable storage. Let us assume that the data limits are an

arbitrarily long period (e.g., a month, a year, etc.) and mass stable storage can store the summarized data within the period. Then, the long period can be divided into multiple time sections for periodic summarization and the size of one local summary can be decided. Figure 1 shows this environment. Independently summarized data (equal in size) are stored in each time section of  $[t_1, t_2], [t_2, t_3], \dots, [t_i, t_{i+1}]$ . All intervals close on the left and open on the right.



**Fig. 1.** Periodic summarization

Let us consider the following problem when using the wavelets technique for this periodic summarization: the local summarization should be more effective so that the sum of local estimation errors by the local wavelet coefficients can be close to global estimation error by the global wavelet coefficients from a single global summarization. That is, if  $[t_1, t_{i+1}]$  is the last time section and  $e_{[t_1, t_{i+1}]}$  is the global estimation error of the time section  $[t_1, t_{i+1}]$ , it is desirable to satisfy the following:  $e_{[t_1, t_{i+1}]} \approx \sum_{i \leq I} e_{[t_i, t_{i+1}]}$ . However, this should be not likely to occur for any other continuous summarizing techniques, so a method to gain much lower overall estimation error (i.e., the sum of all local estimation errors) is needed.

## 2 Our Flexible Approach

### 2.1 Data Based and Query Based Estimation Error

Before introducing the concept of flexible storage allocation, consider the methods for measuring the estimation error by approximating the original data. We classified them into two groups: data based method and query based method. For the data based method, periodic summarization compresses the original data with a lossy-compression scheme such as the wavelet approach, and calculates the absolute difference between the decompressed value and the original data value. Let  $D_k$  be the original  $k$ th data value in a specific time section  $[t_b, t_{i+1})$  and  $D'_k$  be the decompressed data value. The following error measures are defined:

- Absolute error:  $abs\_e_{[t_i, t_{i+1}]} = \sum_k |D_k - D'_k|$
- Relative error:  $rel\_e_{[t_i, t_{i+1}]} = \sum_k (|D_k - D'_k| / |D_k|), D_k \neq 0$ .

The overall estimation error for all the local time sections can also be defined as absolute type ( $abs\_e$ ) or relative type ( $rel\_e$ ), e.g.,  $abs\_e = \sum_i abs\_e_{[t_i, t_{i+1}]}$ .

For the query based method, the estimation error is defined as dependent on the result size of a query. Let  $R_k$  be the actual size of a query  $q_k$  in a specific time section  $[t_b, t_{b+1})$  and let  $R'_k$  be the estimated size of the query. The absolute and relative errors in a specific time section can be defined for the given queries as above. In particular, the  $p$ -norm average error has been defined as the estimation error for the given  $Q$  queries in the reference [6], and we use it here. For  $p > 0$ :

- Absolute error:  $abs\_e_{[t_b, t_{b+1})} = ((\sum_{1 \leq k \leq Q} |R_k - R'_k|^p) / Q)^{1/p}$
- Relative error:  $rel\_e_{[t_b, t_{b+1})} = ((\sum_{1 \leq k \leq Q} (|R_k - R'_k| / R_k)^p) / Q)^{1/p}, R_k > 0.$

For example, for  $p = 1$ , the *1-norm average absolute error* is defined as  $(\sum_{1 \leq k \leq Q} |R_k - R'_k|) / Q$ , and for  $p = 2$ , the *2-norm average absolute error* is defined as  $\sqrt{(\sum_{1 \leq k \leq Q} |R_k - R'_k|^2) / Q}$ .

## 2.2 Flexible Selection of Wavelet Coefficients

In the article [3], we already introduced the necessity of summarizing data stream periodically, using histograms and concept hierarchy besides wavelets. However, the experiments did not show the significant improvement because they used the data based estimation error. Due to page limit we skip the details of our algorithm, and we will concentrate on showing additional experimental results. We refer the reader to the reference [3]. Our basic idea is to exclude the wavelet coefficients less significant to reduce the local estimation error of any time section and utilize the saved storage space from the excluded coefficients for the other time sections.

Table 1 shows the improvement ratio by our flexible approach against the existing fixed approach, and the surplus storage space of our approach. Some real data streams provided at the website [7] were used in our experiments: stock data, EEG measurements of an albin rat, and light measurements from motes. And the following query sets were used for the query based measurement:

$$\{X_i \mid X_i = a\}, \{X_i \mid a \leq X_i \leq b\}, \{X_i \mid (X_i - X_{i-1}) \geq a \text{ } \parallel (X_{i+1} - X_i) \geq a\},$$

the variable  $X_i$  defines an  $i$ th data element from a sensor and the constants  $a$  and  $b$  are a real number.

The surplus storage space of the existing fixed approach is surely zero because of the fixed compressing size for all the local time sections, but our flexible compression can have a surplus space. This surplus space can be used for the later time sections. The improvement ratio under the query based error measurement shows that the correctness of the fixed approach is each 8.8 %, 17.1 %, 15.7 % less than that of our flexible approach. However, the very low improvement ratio under the data based error measurement shows that our flexible approach is more effective when the estimation error depends on given queries.

The query based error measurement is more advantageous to *predefined* queries than *ad hoc* queries, because it keeps more wavelet coefficients significant to given queries (but, note that the selected coefficients can also be relevant to ad hoc queries). The predefined query is one issued before any relevant data has arrived, on the other hand the ad hoc query is one issued after [2].

**Table 1.** Improvement ratio by our flexible Approach against the existing fixed approach

Data Set	Query Based Error Measurement		Data Based Error Measurement	
	Improvement (%)	Surplus Space (byte)	Improvement (%)	Surplus Space (byte)
Stock	8.8	22,604	-3.1	7,580
EEG	17.1	70,808	0	48
Light	15.7	3,148	0.1	6,112

*Improvement* = { (the overall estimation error of the fixed approach) – (the overall estimation error of the flexible approach) } / (the overall estimation error of the fixed approach) × 100 %

The absolute data based measurement and *l*-norm absolute query based measurement were used.

### 3 Conclusions

In this article, we have introduced a periodic data stream summarization for storing as much information about data as possible with lowering the overall estimation error. The proposed method is to adjust the compressing size of each local time section flexibly. Additional experimental results have shown that our flexible approach has lower estimation error than the existing fixed approach, especially in the case of using the query based estimation error. Although the query based estimation error is more advantageous to predefined queries than ad hoc queries, it does not mean that the selected wavelet coefficients cannot be relevant to ad hoc queries at all.

**Acknowledgements.** This study is supported in part by the Second Stage of BK21.

### References

1. A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, W. Hong, “Model-Driven Data Acquisition in Sensor Networks”, Proc. 30th International Conf. on VLDB, Toronto, Canada, pp. 588-599, Sept. 2004.
2. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and Issues in Data Stream Systems”, Proc. the 21th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Madison, USA, pp. 1-16, June 2002.
3. J. Kim and S. Park, “Periodic Streaming Data Reduction Using Flexible Adjustment of Time Section Size”, International Journal of Data Warehousing & Mining, Vol. 1, No. 1, pp. 37-56, Jan. 2005.
4. P. Karras and N. Mamoulis, “One-Pass Wavelet Synopses for Maximum-Error Metrics”, Proc. 31th International Conf. on VLDB, Trondheim, Norway, pp. 421-432, Sept. 2005.
5. R. S. Istepanian, E. Jovanov, and Y. T. Zhang, “Introduction to the special section on M-Health: beyond seamless mobility and global wireless health-care connectivity”, Guest Editorial, IEEE Transactions on Information Technology in Biomedicine, Vol. 8, No. 4, pp. 405-413, Dec. 2004.
6. Y. Matias, J. S. Vitter, and M. Wang, “Dynamic Maintenance of Wavelet-Based Histograms”, Proc. 26th International Conf. on VLDB, Egypt, pp. 101-110, Sept. 2000.
7. Time Series Data Mining Archive. <http://www.cs.ucr.edu/~eamonn/TSDMA/index.html>

# Versioned Relations: Support for Conditional Schema Changes and Schema Versioning

Peter Sune Jørgensen and Michael Böhlen

Faculty of Computer Science, Free University of Bozen-Bolzano, Piazza Domenicani-3-Dominikanerplatz, I-39100 - Bozen-Bolzano, Italy  
`{jorgensen, boehlen}@inf.unibz.it`

**Abstract.** We introduce the versioned relational data model, which allows a user to apply conditional schema changes to a populated database without breaking applications compiled against an existing schema, and without loss of existing data. Our model is based on keeping a history of conditional schema changes, and converting tuples on demand to fit the correct schema in any schema version.

We provide a concrete definition of schema versioning: The ability to specify an operator on any schema version, such that the tuples in the result are unaffected by schema versions created after the specified schema version. Finally, we show that our model supports schema versioning.

## 1 Introduction

Changing the schema of a populated database without breaking existing applications and without loss of existing data remains an open issue. A common ad hoc technique is to add new attributes on demand, and to ignore requests for removal of attributes. This means, if we add an attribute, we pad the existing tuples with nulls to make them fit the new schema. Conversely, since we never remove existing attributes, we must also add nulls to new tuples to make them fit the schema. Consequently, we may break applications, which rely on the existing data and the existing schema.

We develop the versioned relational data model, which lets a user apply schema changes without breaking existing applications. We consider conditional schema changes as first introduced by Jensen and Böhlen [1]. A conditional schema change only changes the schema for the tuples, which satisfy a given condition, e.g., add a SSN attribute to a customer relation, where the condition is that the customer's country is USA.

We introduce a concrete definition of the schema versioning for versioned relations: The ability to specify an operator on any schema version, such that the result is unaffected by schema versions created after the specified schema version. Consequently, if schema versioning is supported, we can change the schema without breaking applications compiled against an existing schema version. Finally, we show that our model supports schema versioning.

## 2 Versioned Relations

A *conditional schema change*  $\delta$  is a 3-tuple:  $\delta = \{\Phi, \mathcal{A}, C\}$ , where  $\Phi$  is either *add* or *remove*,  $\mathcal{A}$  is a set of attributes, and  $C$  is a condition.  $\Phi$  determines if the conditional schema change adds or removes the set of attributes  $\mathcal{A}$ , when the condition  $C$  is satisfied. For example, the conditional schema change  $\{\text{add}, \{\text{workprg.}\}, \text{date} < 01/01/06\}$  adds the attribute *workprg.*, when date  $< 01/01/06$ .

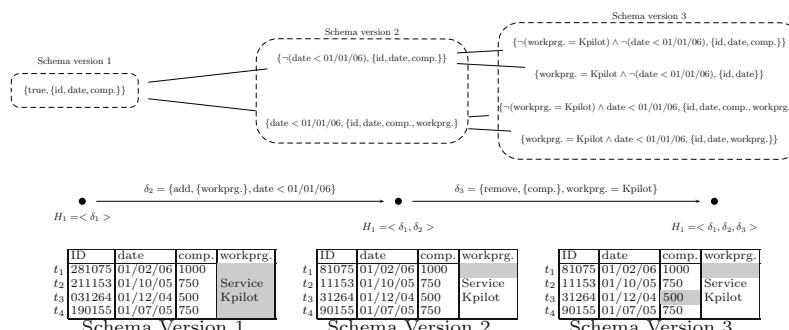
A *conditional schema CS* is a condition-schema pair:  $\text{CS} = \{C, S\}$ . A tuple  $t$  fits a conditional schema  $\{C, S\}$ , if the tuple  $t$  satisfies the condition  $C$ , and if the same attributes occur in the schema  $S$  and in the tuple  $t$ .

*Example 1.* The tuple  $t_1 = \{\text{ID} \rightarrow 281075, \text{date} \rightarrow 01/02/06, \text{comp.} \rightarrow 1000\}$  fits the conditional schema  $\text{CS}_1 = \{\neg(\text{date} < 01/01/06), \{\text{ID}, \text{date}, \text{comp.}\}\}$ .

A *history H* is a list of conditional schema changes  $\langle \delta_1, \dots, \delta_q \rangle$ . A *schema version V* is a set of conditional schemas  $\{\text{CS}_1, \dots, \text{CS}_z\}$ . A *versioned relation schema VS(H)* is a list of schema versions  $\langle V_1, \dots, V_m \rangle$ . When we apply a conditional schema change to a versioned relation schema  $\text{VS}(H)$ : We add the conditional schema change to the history  $H$ , and we add a schema version to the versioned relation schema consisting of 2 conditional schemas for every conditional schema in the previous schema version: (1) a conditional schema, where the schema and the condition is changed according to the conditional schema change, and (2) a conditional schema, where we add the negation of the condition of the conditional schema change.

*Example 2.* We store information about unemployed citizens in the versioned relation *Unemp*. In schema version 1, we store citizens with an *ID*, a *date* of unemployment, and the *compensation* they receive. In schema version 2, we add a *workprogram* for citizens, who have been unemployed since before 01/01/06. In schema version 3, we remove the *compensation* for citizens in the Kpilot workprogram. This yields the history  $H = \langle \delta_1, \delta_2, \delta_3 \rangle$ , where  $\delta_1 = \{\text{add}, \{\text{ID}, \text{date}, \text{comp.}\}, \text{true}\}$ ,  $\delta_2 = \{\text{add}, \{\text{workprg.}\}, \text{date} < 01/01/06\}$ , and  $\delta_3 = \{\text{remove}, \{\text{comp.}\}, \text{workprg.} = \text{Kpilot}\}$ . The versioned relation schema  $\text{VS}(H)$  is illustrated in Fig. 1.

A *versioned relation R(H)* is a set of tuples, where every tuple fits a conditional schema in the versioned relation schema  $\text{VS}(H)$ . We determine the *correct schema* for a tuple



**Fig. 1.** The versioned relation schema  $\text{VS}(\langle \delta_1, \delta_2, \delta_3 \rangle)$  and the tuples of the versioned relation *Unemp(H)* converted into their correct schema in each schema version

$t$  in schema version  $v$  of the versioned relation schema  $VS(H)$  by applying the first  $v$  conditional schema changes in the history  $H$  to an empty schema, where we use the tuple  $t$  to evaluate the condition of each conditional schema change.

**Lemma 1.** *Applying a conditional schema change to a versioned relation schema does not change the correct schema for a tuple in an existing schema version.*

We use the function  $\text{convert}(t, v, \Gamma, H)$  to *convert* the tuple  $t$  to fit the correct schema in schema version  $v$  of the versioned relation schema  $VS(H)$ . In the conversion we remove attributes, which do not occur in the correct schema, and we add attribute values, which are missing with *mismatch resolutions*. A mismatch resolution  $\gamma$  is an attribute-tuple mapping pair  $\{A, t \rightarrow x\}$ , where  $t \rightarrow x$  is a mapping from a tuple  $t$  to a value  $x$  for the attribute  $A$ .

*Example 3.* Fig. 11 illustrates the tuples of the versioned relation  $Unemp(H)$  converted to their the correct schema in each of the 3 schema versions. Note that, an attribute is grey, if it is missing in that schema version, and an attribute without a value means the value is missing in that schema version.

### 3 Algebra for Versioned Relations and Schema Versioning

In Table 11 we define the following operators: Selection ( $\sigma^{v, \Gamma}$ ), projection ( $\pi^{v, \Gamma}$ ), union ( $\cup^{v, w, \Gamma}$ ), and difference ( $-^{v, w, \Gamma}$ ), where  $v$  and  $w$  are version numbers, and  $\Gamma$  is a set of mismatch resolutions.

*Example 4.* Here we show an example operation, where a legacy application adds a tuple to the versioned relation  $Unemp(H)$  using schema version 2

$$\text{Unemp} \cup^{2,1,\emptyset} \{\text{ID} \rightarrow 130977, \text{date} \rightarrow 03/11/04, \text{comp.} \rightarrow 475, \text{workprg} \rightarrow \text{Kpilot}\}$$

This succeeds, since the tuple fits the correct schema in schema version 2.

**Table 1.** The algebra for versioned relations

$\sigma^{v, \Gamma}[P](R_1(H_1)) = R_2(H_1)$	$R_2 = \{t \mid t \in R_1 \wedge P(\text{convert}(t, v, \Gamma, H_2))\}$
$R_1(H_1) \cup^{v, w, \Gamma} R_2(H_2) = R_3(H_1)$	$R_3 = R_1 \cup \{\text{convert}(t, w, \Gamma, H_2) \mid t \in R_2 \wedge \text{convert}(t, w, \Gamma, H_2) = \text{convert}(\text{convert}(t, w, \Gamma, H_2), v, \Gamma, H_1)\}$
$R_1(H_1) -^{v, w, \Gamma} R_2(H_2) = R_3(H_1)$	$R_3 = \{t_1 \mid t_1 \in R_1 \wedge \forall t_2 : (t_2 \in R_2 \Rightarrow \text{convert}(t_1, v, \Gamma, H_1) \neq \text{convert}(t_2, w, \Gamma, H_2))\}$
$\pi^{v, \Gamma}[\{A_1, \dots, A_n\}](R_1(H_1)) = R_2(H_2)$	$R_2 = \{\{A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n\} \mid t \in R_1 \wedge t' = \text{convert}(t, v, \Gamma, H_1) \wedge (A_1 \rightarrow x_1) \in t' \wedge \dots \wedge (A_n \rightarrow x_n) \in t'\}$ $H_2 = < \{\text{add}, \{A_1, \dots, A_n\}, \text{true}\} >$

**Definition 1. (Schema versioning)** An operator supports schema versioning, when it can be specified on all schema versions of a versioned relation, such that the tuples in the result of the operator specified on schema version  $v$  are unaffected by schema versions created after schema version  $v$ .

*Example 5.* Let  $v$  be a version number,  $t$  be a tuple,  $H$  be a history,  $R(H)$  be a versioned relation. Then the unary operator  $\mathcal{O}$  supports schema versioning, iff the following constraint is satisfied  $\forall v : (1 \leq v \leq |H| \wedge t \in \mathcal{O}^v(R(H)) \Leftrightarrow \mathcal{O}^v(\delta(R(H))))$ .

**Theorem 1.** *The operators in the algebra for versioned relations support schema versioning.*

*Proof.* The correct schema for a tuple  $t$  in schema version  $v$  is unaffected by schema versions created subsequently (cf. Lemma 1).

## 4 Related Work

Jensen et al.<sup>[2]</sup> provide an abstract definition of schema versioning as the ability to query through user-defined interfaces. We provide a concrete definition of schema versioning for versioned relations: The ability to specify operators on any schema version, such that the tuples in the result are unaffected by schema versions created after the specified schema version. In contrast with the abstract consensus definition, we can show that our model satisfies the concrete definition.

Jensen and Böhlen<sup>[1]</sup> introduced the concept of a conditional schema change, and proposed a data model with support for conditional schema changes without loss of existing data. Their model does not ensure that applications relying on the existing schema will continue to function after a schema change.

Jensen and Böhlen<sup>[3]</sup> describe how a history of the conditional schema changes can be used to classify tuples as legacy, current, or invalid tuples. We make full use of the history of conditional schema changes to determine the correct schema for a tuple in any schema version.

## 5 Conclusion

We have provided a concrete definition of schema versioning, and we have defined the versioned relational data model with a set of algebraic operators. We have shown how our model supports conditional schema changes and schema versioning.

## References

1. Jensen, O.G., Böhlen, M.H.: Evolving relations. In: FMLDO. (2000) 115–132
2. Jensen, C.S., Clifford, J., Elmasri, R., Gadia, S.K., Hayes, P.J., Jajodia, S.: A consensus glossary of temporal database concepts. SIGMOD Record **23** (1994) 52–64
3. Jensen, O.G., Böhlen, M.H.: Current, legacy, and invalid tuples in conditionally evolving databases. In: ADVIS. (2002) 65–82

# Compatibility Analysis and Mediation-Aided Composition for BPEL Services

Wei Tan<sup>1</sup>, Fangyan Rao<sup>2</sup>, Yushun Fan<sup>1</sup>, and Jun Zhu<sup>2</sup>

<sup>1</sup> Department of Automation, Tsinghua University, 100084 Beijing, P.R. China

<sup>2</sup> IBM China Research Lab, 100094 Beijing, P.R. China

tanwei@mails.tsinghua.edu.cn, raofy@cn.ibm.com,  
fanyus@tsinghua.edu.cn, zhujun@cn.ibm.com

**Abstract.** In Service Oriented Architecture (SOA), the need for inter-service compatibility analysis has gone beyond what existing service composition/verification approaches can handle. Given two services whose interface invocation constraints are described by Business Process Execution Language for Web Services (BPEL4WS, or BPEL), we analyze their compatibility and adopt mediation as a light weight approach, to make partial compatible services work together more adaptively, without changing their internal logic. We transform BPEL into service workflow net which is a kind of colored Petri net. Based on this formalism we first analyze the compatibility of two services, and then devise an approach to check whether there exists any message mediation so that their mediation-aided composition will not violate the constraints imposed by either side. Later the method for mediation generation is also introduced. Our approach is validated through a real life case and further research directions are pointed out.

## 1 Introduction

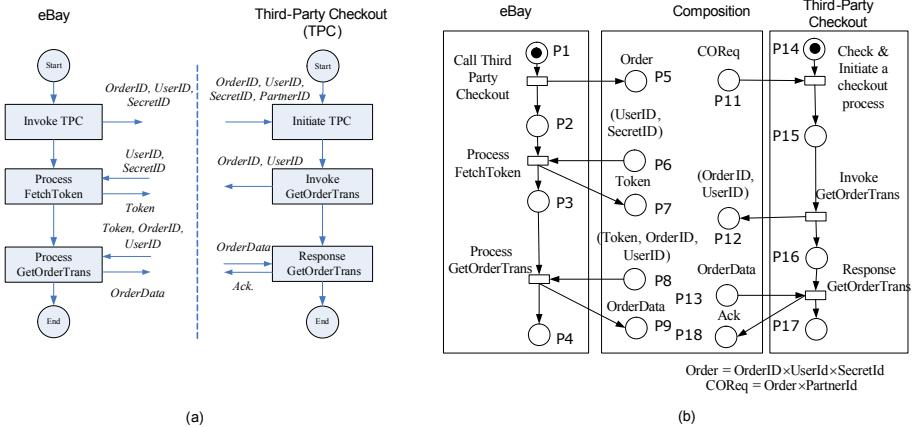
We observed that *partial compatibility* is a common phenomenon in real-life web service composition, that is, two (or more) web services provide complementary functionality; however, their interaction patterns do not necessarily fit exactly so that they cannot be directly composed. Current research in web service composition pays little attention to the partial compatibility issue.

Recently, the mediation approach is attracting more attention [1-3]. Mediation (or mediator) wraps heterogeneous services so that they can appear as homogeneous and therefore is easier to be integrated.

Compared to existing work in service compatibility analysis, our contributions are:

1. We use state space based method to check the existence of mediator rigorously.
2. We propose the guidance to generate mediator to glue two services.

Our motivating scenario comes from the composition of eBay and a third-party checkout service [4]. eBay, an online auction and shopping service provider, allows a third party to handle a seller's checkout processes on eBay. In our scenario, we are to integrate the eBay service with some other third-party checkout service so that buyers can bid on eBay and checkout on another website. Fig. 1 (a) illustrates the internal processes of these two services. From the figure we can observe that the messages



**Fig. 1.** (a) eBay and third party checkout services (b) SWF-Nets of two services

content and sequence of both services do not fit exactly so that they could not be directly composed. We will use this example to illustrate our approach in this paper.

## 2 Solution Approach

Our solution is based on formal model. First, we transform two BPEL services which are to be composed to SWF-nets (which is a kind of colored Petri net), then we verify whether the two services are directly composable, if the answer is *no*, we require data mapping information. Then we'll use data mapping to build Communicating Reachability Graph (CRG) to verify whether there exists a mediator to glue the two services, and if the answer is *yes*, we generate the mediator.

Fig. 1 (b) depicts the result of transforming two BPEL service in Fig. 1(a) into SWF-nets.

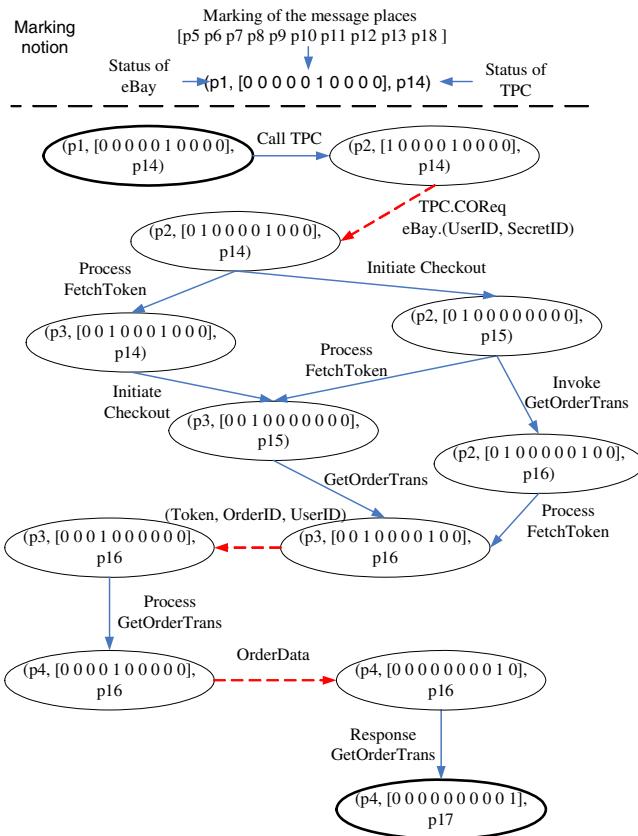
Data mapping is to define rules to relate (syntactically/semantically) equivalent elements of two messages so that two interfaces which belong to different services can be linked. By specifying the output message of one interface as the input message of another one, two services can be composed. Data mapping can be at message level, parts level or element level. In eBay example, we have the following data mapping, as Table 1 shows, and this data mapping table is given manually.

In order to check whether there exists a mediator to glue two partial compatible services, we introduce the concept of Communicating Reachability Graph (CRG). The basic idea of CRG is to construct the reachability graph of two services concurrently, using data mapping as the communication mechanism. That is, when the source data is ready, their target should be informed.

Given eBay service  $SN_1$ , TPC service  $SN_2$  in Fig. 1 (b), and the data mapping  $I$  in Table 1, we can derive  $CRG(SN_1, SN_2, I)$ , as Fig. 2 shows. In Fig. 2, the operation edges are denoted with solid lines, and the names of the operation transition are labeled on the lines (for example, the transition *Call TPC*). The mediation edges are denoted with

**Table 1.** Data mapping table

Source	Target
eBay.Order	TPC.COReq.Order
“eBay”	TPC.COReq.PartnerId
eBay.Order.(UserID, SercretID)	eBay.(UserID, SercretID)
TPC. (OrderID, UserID)	eBay.(Token, OrderID, UserID).(OrderID, UserID)
eBay.Token	eBay. (Token, OrderID, UserID).Token
eBay.OrderData	TPC.OrderData

**Fig. 2.** The Communicating Reachability Graph of eBay and TPC service

dashed lines, and the data obtained by mediation are labeled on the dashed lines (for example, the transition *OrderData*).

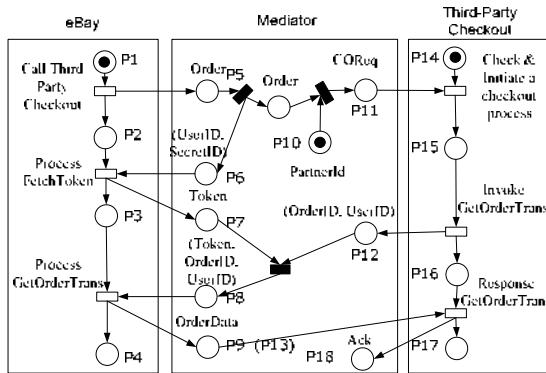
**Proposition 1.** Given two SWF-nets  $SN_1$  and  $SN_2$ , and data mapping  $I$  (which is complete and accurate), there exists a mediation  $MED$  w.r.t.  $I$ , and  $SN_1$  and  $SN_2$  can be composed via  $MED$  iff  $CRG(SN_1, SN_2, I)$  is well-formed, that is,

- 1) For each reachable marking (starting at  $M_0$ ), the final marking  $M_e$  is coverable.
- 2) For each reachable marking  $M$  such that  $M \geq M_e$  holds, for  $\forall p$  s.t.  $M(p) > M_e(p)$ ,  $p \in MP_1 \cup MP_2$ .

Due to the limitation of space, we omit the proof details of this proposition. We can easily verify that in Fig. 2, CRG( $SN_1, SN_2, I$ ) is well-formed. Therefore we claim that eBay and TPC can be composed with the aid of mediator.

Now we'll give the method to build the mediator between message places of  $SN_1$  and  $SN_2$ , if we can judge the existence of mediator by verifying that the CRG is well-formed.

A mediator between eBay service and TPC service can be generated according to the information we obtained in CRG, as Fig. 3 illustrates. The mediator transitions are denoted with black rectangles to differentiate them with operation transitions belonging to eBay and TPC services.



**Fig. 3.** Mediator between eBay and TPC

### 3 Conclusion

In future work, we plan to find more real life cases to validate our idea; at the same time, we're going to make further investigations on the properties of data mapping and its influence on mediation existence and generation.

### References

1. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2): 113–137, 2002.
2. B. Benatallah, et al. Developing Adapters for Web Services Integration. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, 2005.
3. D.M. Yellin and R.E. Strom. Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2): 292-333, 1997.
4. eBay. Third Party Checkout. 2006. [http://developer.ebay.com/DevZone/XML/docs/WebHelp/Checkout-Third\\_Party\\_Checkout.html](http://developer.ebay.com/DevZone/XML/docs/WebHelp/Checkout-Third_Party_Checkout.html)

# Efficient Reasoning About XFDs with Pre-image Semantics

Sven Hartmann, Sebastian Link, and Thu Trinh

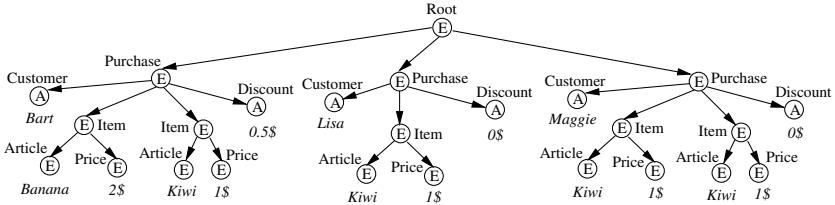
Information Science Research Centre, Department of Information Systems  
Massey University, Palmerston North, New Zealand  
`{s.hartmann, s.link, t.trinh}@massey.ac.nz`

**Abstract.** The study of integrity constraints has been identified as one of the major challenges in XML database research. The main difficulty is finding a balance between the expressiveness and the existence of automated reasoning tools. We investigate a previous proposal for functional dependencies in XML (XFDs) that is based on homomorphisms between data trees and schema trees. We demonstrate that reasoning about our XFDs is well-founded. We provide a finite axiomatisation and show that their implication is equivalent to the logical implication of propositional Horn clauses and thus decidable in time linear in the size of the constraints. Hence, our XFDs do not only capture valuable semantic information but also permit efficient automated reasoning support.

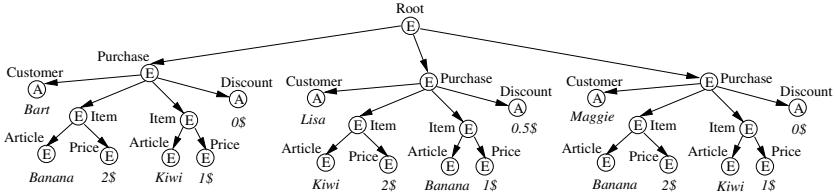
## 1 Introduction

The importance of XML integrity constraints is due to a wide range of applications ranging from schema design, query optimisation, efficient storing and updating, data exchange and integration, to data cleaning [3]. Several classes of integrity constraints have been defined for XML including functional dependencies [1456891011]. While there is a well-accepted single concept for the notion of functional dependency in relational databases the complex nature of XML data has resulted in various proposals for XFDs that deviate in their expressiveness but are all justified as they naturally occur in XML data.

For an example consider the XML data tree in Figure 1 that stores simple purchase profiles showing customers, the items they bought (an item is a pair consisting of an article and its price) and the discount received for the purchase. In the data tree the same articles have the same price. This observation is likely to be called a functional dependency between the article and its price. In Figure 2, this functional dependency is no longer valid. Still, the data stored in this tree is not independent from one another: whenever two customers have purchased all the same items then they both receive the same discount. That is, the set of items purchased functionally determines the discount. This dependency does not occur just accidentally but captures important semantic information that should be satisfied by every legal data tree of this form. *Lisa* might have received a discount of 0.5\$ since *Kiwis* for the price of 2\$ were on special.



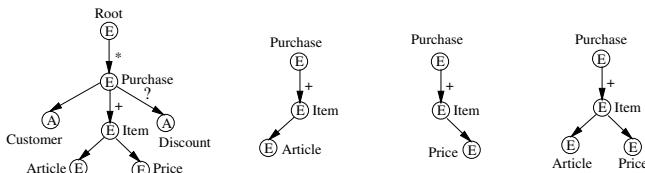
**Fig. 1.** XML data tree exhibiting some functional dependency



**Fig. 2.** Another XML data tree exhibiting another kind of functional dependency

The majority of proposals has considered the first kind of XFDs [16, 9] which is reminiscent of earlier research on path-based dependencies in semantic and object-oriented data models, while this paper studies the second kind [4, 10]. We use the simple XML graph model from [45]. An *XML tree* is a rooted tree  $T$  with node set  $V_T$ , arc set  $A_T$ , root  $r_T$ , and mappings  $\text{name} : V_G \rightarrow \text{Names}$  and  $\text{kind} : V_G \rightarrow \{E, A\}$ . The symbols  $E$  and  $A$  indicate elements and attributes. A *data tree* is an XML tree  $T'$  with string values assigned to its leaves. Two data trees  $T'$  and  $T$  are *value-equal* if there is a value-preserving isomorphism between them. A *schema tree* is an XML tree  $T$  where no two siblings have the same name and kind, and with frequencies assigned to its arcs.

A *v-walk* of an XML tree  $T$  is a directed path from a fixed node  $v$  to some leaf of  $T$ . A *v-subgraph* of  $T$  is the union of  $v$ -walks of  $T$ . Clearly, a  $v$ -subgraph is an XML tree again. The empty  $v$ -subgraph is denoted by  $\emptyset_{T,v}$ . The *total v-subgraph*  $T(v)$  is the union of all  $v$ -walks of  $T$ . Consider two XML trees  $T'$  and  $T$  with a homomorphism between them. An  $r_{T'}$ -subgraph  $U'$  of  $T'$  is a *subcopy* of  $T$  if



**Fig. 3.** A schema tree with arc labels for the frequencies, and three of its  $v_{\text{Purchase}}$ -subgraphs: the  $v_{\text{Purchase}}$ -walks  $[\text{Article}]$  and  $[\text{Price}]$ , and their union  $[\text{Article}, \text{Price}]$  (for convenience, we use an example where walks can be identified by their leaf names)

$U'$  is isomorphic to some  $r_T$ -subgraph  $U$  of  $T$ , and an *almost-copy* of  $T$  if it is maximal with this property. Given an  $r_T$ -subgraph  $U$  of  $T$ , the *projection* of  $T'$  to  $U$  is the union of all subcopies of  $U$  in  $T'$ , and denoted by  $T'|_U$ .

In relational databases, a functional dependency  $X \rightarrow Y$  is satisfied if and only if any two tuples that agree on their projections to  $X$  also agree on their projections to  $Y$ . Surprisingly, it is not that obvious how to translate the concept of functional dependency to XML. Most importantly, one has to decide what the tuples in an XML data tree should be. Arenas/Libkin [1] suggested to consider almost-copies of a schema tree  $T$  in a  $T$ -compatible data tree  $T'$  as tree-tuples. Almost-copies are of interest as  $T'$  does not necessarily contain copies of  $T$ . XFDs of this kind have been studied in detail, e.g., in [1, 5, 8, 9, 11].

## 2 Deciding Implication of XFDs Based on Pre-images

The homomorphism between a  $T$ -compatible data tree  $T'$  and a schema tree  $T$  induces a mapping of the total subgraphs of  $T'$  to the total subgraphs of  $T$ . For a fixed node  $v$  of  $T$ , the pre-images of the total  $v$ -subgraph  $T(v)$  are just the total subgraphs rooted at the pre-images of the node  $v$  in  $T'$ . In [4] we suggested to consider pre-images as tree-tuples and gave natural examples for such XFDs. Given  $T$  and  $v$ , a *functional dependency* (*XFD*,  $v$ -*XFD*) is an expression  $v : \mathcal{X} \rightarrow \mathcal{Y}$  where  $\mathcal{X}$  and  $\mathcal{Y}$  are non-empty sets of  $v$ -subgraphs of  $T$ .  $T'$  satisfies  $v : \mathcal{X} \rightarrow \mathcal{Y}$  if and only if for any two pre-images  $W_1$  and  $W_2$  of  $T(v)$  in  $T'$  the projections  $W_1|_Y$  and  $W_2|_Y$  are value-equal for all  $Y \in \mathcal{Y}$  whenever the projections  $W_1|_X$  and  $W_2|_X$  are value-equal for all  $X \in \mathcal{X}$ .

For example, the data tree in Figure 1 satisfies  $v_{Purchase} : \llbracket Article, Price \rrbracket \rightarrow \llbracket Discount \rrbracket$ . This can be checked by inspecting the three tree-tuples, that is, the total subgraphs rooted at the three pre-images of  $v_{Purchase}$ . The data tree in Figure 2 satisfies the same XFD. This is noteworthy as the latter data tree does not satisfy the XFD  $\llbracket Article, Price \rrbracket \rightarrow \llbracket Discount \rrbracket$  when based on almost-copies as tree-tuples. Note that the definition of  $v$ -XFDs should not be simplified to expressions  $v : X \rightarrow Y$  with single  $v$ -subgraphs  $X$  and  $Y$  as this causes a loss of expressiveness, e.g., the XFDs  $v_{Purchase} : \llbracket Article, Price \rrbracket \rightarrow \llbracket Discount \rrbracket$  and  $v_{Purchase} : \llbracket Article \rrbracket, \llbracket Price \rrbracket \rightarrow \llbracket Discount \rrbracket$  are different from one another. In fact, the data tree in Figure 2 satisfies the former XFD, but not the latter one.

**Theorem 1.** *The inference rules below form a sound and complete set of inference rules for the implication of  $v$ -XFDs:*

$$\begin{array}{c}
 \dfrac{}{v : \emptyset_{T,v} \rightarrow T(v)}^{v \text{ simple}} \quad \dfrac{v : X \rightarrow Y}{\text{of } X} \quad \dfrac{v : X, Y \rightarrow X \sqcup Y}{X, Y \text{ reconcilable}} \\
 (\text{uniqueness}) \qquad \qquad \qquad (\text{subgraph}) \qquad \qquad \qquad (\text{join}) \\
 \hline
 \dfrac{}{v : \mathcal{X} \rightarrow \mathcal{Y}}^{\mathcal{Y} \subseteq \mathcal{X}} \quad \dfrac{v : \mathcal{X} \rightarrow \mathcal{X} \cup \mathcal{Y}}{\text{(extension)}} \quad \dfrac{v : \mathcal{X} \rightarrow \mathcal{Y}, v : \mathcal{Y} \rightarrow \mathcal{Z}}{v : \mathcal{X} \rightarrow \mathcal{Z}}
 \\ (\text{reflexivity}) \qquad \qquad \qquad (\text{extension}) \qquad \qquad \qquad (\text{transitivity})
 \end{array}$$

The uniqueness axiom states that if the path from the root to the node  $v$  in the schema tree  $T$  is *simple*, i.e., does only contain arcs of frequency ? or 1,

then a  $T$ -compatible data tree  $T'$  has at most one pre-image of  $T(v)$ . The join axiom gives a sufficient (and also necessary) condition when the projections of a pre-image  $W$  of  $T(v)$  on two  $v$ -subgraphs  $X$  and  $Y$  uniquely determine the projection on their union  $X \sqcup Y$ . Two  $v$ -subgraphs  $X, Y$  are called *reconcilable* if whenever  $X$  and  $Y$  share some arc  $(u, w)$  of frequency other than ? or 1, then  $X$  contains the total  $w$ -subtree of  $Y$  or  $Y$  contains the total  $w$ -subtree of  $X$ .

In the sequel we discuss how to decide implication efficiently. Let  $T$  be a schema tree, and  $v$  a node of  $T$ . The set  $\mathcal{B}(v)$  of *essential subgraphs* is defined as the smallest set of  $v$ -subgraphs of  $T$  such that every  $v$ -walk of  $T$  belongs to  $\mathcal{B}(v)$  and if  $X, Y \in \mathcal{B}(v)$  are not reconcilable then  $X \sqcup Y \in \mathcal{B}(v)$ . Note that two pre-images that coincide on the projections to all members of  $\mathcal{B}(v)$  must be value-equal, and  $\mathcal{B}(v)$  is the smallest set with this property. For a set  $\mathcal{X}$  of  $v$ -subgraphs of  $T$  let  $\vartheta(\mathcal{X})$  contain all the essential subgraphs in  $\mathcal{B}(v)$  that are subgraphs of some member of  $\mathcal{X}$  and are maximal with respect to this property, i.e.,  $\vartheta(\mathcal{X}) = \max\{Y \in \mathcal{B}(v) : Y \text{ is } v\text{-subgraph of } X \text{ for some } X \in \mathcal{X}\}$ . A  $T$ -compatible XML data tree  $T'$  satisfies the XFD  $v : \mathcal{X} \rightarrow \mathcal{Y}$  if and only if  $T'$  satisfies the XFD  $v : \vartheta(\mathcal{X}) \rightarrow \vartheta(\mathcal{Y})$ . We may therefore assume without loss of generality that every XFD  $v : \mathcal{X} \rightarrow \mathcal{Y}$  is of the form  $\mathcal{X} = \vartheta(\mathcal{X})$  and  $\mathcal{Y} = \vartheta(\mathcal{Y})$ .

Now we establish a correspondence between the implication of XFDs and the logical implication of propositional Horn clauses. Let  $\varphi : \mathcal{B}(v) \rightarrow \mathcal{V}$  be a mapping that assigns propositional variables to the  $v$ -subgraphs of  $T$ . If  $\sigma$  is an XFD  $v : \{X_1, \dots, X_k\} \rightarrow \{Y_1, \dots, Y_n\}$  on  $T$ , then let  $\Pi_\sigma$  be the set of the following  $n$  Horn clauses:  $\neg\varphi(X_1) \vee \dots \vee \neg\varphi(X_k) \vee \varphi(Y_1), \dots, \neg\varphi(X_1) \vee \dots \vee \neg\varphi(X_k) \vee \varphi(Y_n)$ . If  $\Sigma$  is a set of  $v$ -XFDs on  $T$ , then let  $\Pi_\Sigma$  be the union of the sets  $\Pi_\sigma$ ,  $\sigma \in \Sigma$ . Further, the structure of  $\mathcal{B}(v)$  can be encoded by the set  $\Pi_T = \{\neg\varphi(U) \vee \varphi(W) : U, W \in \mathcal{B}(v), U \text{ covers } W\}$ , where a  $v$ -subgraph  $U$  is said to *cover* a  $v$ -subgraph  $W$  if  $U$  is the union of  $W$  and just one additional  $v$ -walk of  $T$ .

**Theorem 2.** *Let  $\Sigma \cup \{\sigma\}$  be a set of  $v$ -XFDs on  $T$ .  $\Sigma$  implies  $\sigma$  if and only if  $\Pi_\Sigma \cup \Pi_T$  logically implies  $\Pi_\sigma$ .*

**Corollary 3.** *The problem whether  $\Sigma$  implies  $\sigma$  can be decided in time linear in the total number of essential subgraphs in  $\Sigma$ .*

The corollary follows straight from the linear time decidability for the implication of propositional Horn clauses [2]. Thus, XFDs based on pre-images do not only occur naturally in XML data but enjoy well-founded reasoning techniques that can be implemented efficiently for native XML data management. This is in contrast to many other classes of XML constraints [3].

## References

1. M. Arenas, L. Libkin. A normal form for XML documents. *ACM ToDS* 29, 2004.
2. W. Dowling, J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming* 1, 1984.
3. W. Fan. XML constraints. *DEXA Workshops* 2005.

4. S. Hartmann, S. Link. More functional dependencies for XML. *ADBIS* 2003, LNCS 2798.
5. S. Hartmann, T. Trinh. Axiomatising functional dependencies for XML with frequencies. *FoIKS* 2006, LNCS 3861.
6. M. Lee, T. Ling, W. Low. Designing functional dependencies for XML. *EDBT* 2002, LNCS 2287.
7. M. Nicola, B. van den Linden. Native XML support in DB2. *VLDB* 2005.
8. M. Vincent, J. Liu. Completeness and decidability properties for functional dependencies in XML. CoRR cs.DB/0301017, 2003.
9. M. Vincent, J. Liu, C. Liu. Strong functional dependencies and their application to normal forms in XML. *ACM ToDS* 29, 2004.
10. J. Wang. A comparative study of functional dependencies for XML. *APWeb* 2005, LNCS 3399.
11. J. Wang, R. Topor. Removing XML data redundancies using functional and equality-generating dependencies. *ADC* 2005, CRPIT 39.

# Context RBAC/MAC Access Control for Ubiquitous Environment\*

Kyu Il Kim, Hyuk Jin Ko, Hyun Sik Hwang, and Ung Mo Kim

Department of Computer Engineering, Sungkyunkwan University,  
300 Chunchun-dong, Jangan-gu, Suwon,  
Gyeonggi-do 440-746, Republic of Korea  
{kisado, hiko, hhs486, umkim}@ece.skku.ac.kr

**Abstract.** Ubiquitous environment that is omnipresent is existent everywhere or seems to be always present. Such an environment is a next generation paradigm in which many invisible computers are integrated into background of our lives. However, it requires more secure technologies to protect privacy because user may access information without time and space restriction. In this paper, we propose the mechanism that a user is able to automatically access to resource by means of context aware on ubiquitous computing environments. For this purpose, we exploit Role-Based Access Control (RBAC) and Mandatory Access Control (MAC) policies, and defines extended context rules. We also provide an advanced security authorization mechanism and show how to securely preserve properties despite of dynamic change of access control privilege.

**Keywords:** Access Control, RBAC, MAC.

## 1 Introduction

In ubiquitous environment, connected devices can be aware of the status of the users and provide information to the users automatically at any time and from anywhere. This provides with convenience which transcends time and space. But this can cause problems such as a privacy exposure. A mistake can be abused immediately by the criminals. And a small error of the system can lead to big confusion. Security techniques for a ubiquitous computing include access control, user certification and security protocol. This research focuses on privacy access control. Privacy [9] is the ability of an individual or group to keep their lives and personal affairs out of the public view, or to control the flow of information about themselves. The problem to happen in a ubiquitous environment is as follows. For example, we assume if the car moves to the garage when the car is out of order in the road. In case of existing approach give the driver role to user. And

---

\* This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment).

it sends role information to the garage. But driver role can sends the garage to sensitive driver career, accident, drunken driving career and driver of personal as well as general information such as name, address, cars kind and car year. Even a few privacy violations could lead to user distrust and abandonment of context-aware systems and to lost opportunities for great enhancements. The current access control is based on a static infrastructure and is not suitable for context access control or privacy control. The traditional system technique for data access is controlled by access control based static security policy. In this paper, we propose the technique to maintain the confidentiality and the integrity about the private information based on ubiquitous computing. We design the mechanism to apply existing RBAC/MAC to context rule so that it is suitable in ubiquitous environment. The paper is organized as follows: Section2 describes related work in the field of context-aware security. Section3 defines context rules and Section4 presents the secure context RBAC/MAC architecture. Section5 discusses context-aware policy. Finally, Section6 presents conclusions.

## 2 Related Works

In this section, we briefly highlight describes several existing access models influencing work, using environment roles[12] and context-aware access control models. Context-based security has already been applied in various settings. Traditional RBAC is discussed in [8,15,16]. A role is a grouping mechanism used to categorize subjects based on various properties. Individual users in the RBAC model are called subjects. A subject can use any role that it can enter. Each subject has an authorized role set which consists of all roles that the subject is permitted to enter. This paper provides a more versatile and more expressive framework that incorporates the use of context rules, privacy control, and expanded RBAC. Environment roles are really one component in a Generalized Role-Based Access Control Model(GRBAC) [11]. GRBAC is a highly expressive easy-to-use access control model designed with two major goals in mind: flexibility and simplicity. GRBAC is flexible because it provides a policy around subject, objects, environmental conditions, or a combination of all three. In addition, GRBAC is a very simple model. It achieves its goal of flexibility in policy design, using a single general grouping strategy. In GRBAC, access policies are defined by subject roles, environment roles and object roles. In the Web Services area, several mechanisms for controlling access to web services have been proposed. An XML access control language (XACL) for web services has been discussed by Hada and Kudo [6]. XACL does not support roles and does not handle context information. The OASIS eXtensible Access Control Markup Language (XACML) specification is based on an extension of XML to define access control specifications that support notions similar to context based privilege assignments [7]. However, it does not directly support the notion of roles. We present a framework for context RBAC/MAC access control to support the above problems.

### 3 Secure Context RBAC/MAC Model

Our model is a context-aware access control framework for the design and privacy in ubiquitous environment. In this section, we define Context RBAC/MAC model which enables privacy control in context-aware system. Traditional RBAC is very useful. However, it suffers from subject-centric limitations that restrict the policy designer to a subject-oriented viewpoint. That is, the RBAC leaks the strength of security because constraints are only applied to the subject. The access decision is based entirely on the permissions associated with the privacy control, in order to enhance security. RBAC cannot support time-dependent access control, so that, for example, subjects can only access object O between 10:00~18:00 on weekdays. So RBAC cannot easily support context based access control. New model distinguishes two different mechanisms. Context Rules support user situation by time, location, or other contextual information that is relevant to access control. It is flexible enough to support policies that make use of security relevant context rules to control access to objects. As shown in Fig.1, our model provides integration mechanism for mandatory and uses role-based access control. Role assignment of subject doesn't be assigned the administrator but role be assigned automatically by context rules in ubiquitous environment.

#### 3.1 Context Rules

In terms of context, the proposed model describes the properties and structure of context information. In the realm of ubiquitous computing, "context" refers to any information of a particular circumstance, object, or condition surrounding a user considered relevant to the interaction to the user and ubiquitous computing environment. This section defines the context, context type, and context expression terms.

**Definition 1.** (*Context*) Subject's context information(CI) (e.g. Location, Time, Environment, etc.).

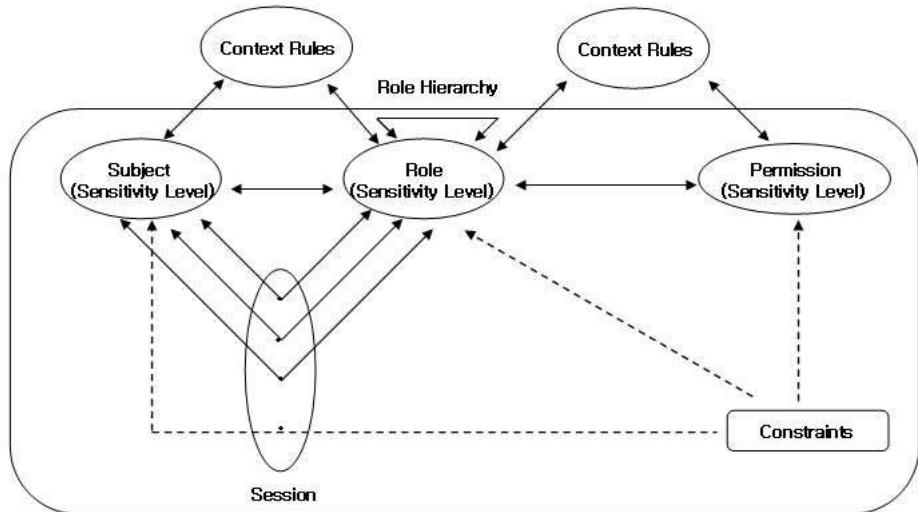
$$CI = c_1, c_2, c_3, c_4, \dots, c_m, \text{ where } c_m \in CI$$

There are different types of contexts can be used by application. For example Location contexts, information context, personal contexts, social contexts, and system contexts, etc. Context type can be formally defined as follow:

**Definition 2.** (*Context Type*) A Context type is a pair  $(ct\_id, \text{attribute})$  and attribute compose a triple  $CTA = [attr\_name, attr\_domain, attr\_value]$ .

Example 1. The following is example of context type.

```
[Location,(University, string, SungKyunKwan)]
[Time,(morning,integer, 09:00)]
[Environment, (weather, string,fine)]
```



**Fig. 1.** Secure Context RBAC/MAC Model

The context expression defines conjunction, disjunction, and negation as operation.

**Definition 3.** (*Context Expression*)  $user\_id \in U, C_i C_j \in CE$  then  $user\_id(C_i \wedge C_j)$  are context expressions.

Example 2. The following are example of context expression.

Bob [Location(house, string, floor)  $\wedge$  Time(evening, integer, 18:00)  
 $\wedge$  Information(action, string, TV seeing)]

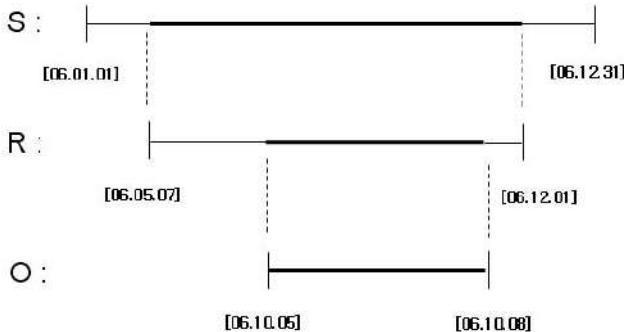
### 3.2 Lifetime and Time Constraint

This section defines about Lifetime and Time constraints. Lifetime is the time required between each subject, role and permission creation and completion.

**Definition 4.** (*A lifetime*)  $LT$ , is a time interval with start time( $st$ ) and end time( $et$ )  $[st, et]$ ,  $et > st$ ;  $st/et$ (year, month, day, hour, min, sec)  $LTs \triangleright Y$  means  $X.st \leq Y.et$  if  $X \cap Y = \emptyset$ ,  $ET \leq ST$ .

**Definition 5.** (*Time Constraint*)  $TC$  each property executes an object according to lifetime, indicating a condition that can be refused. The Time constraint executes an object, the lifetime of each property must overlap.

**Definition 6.** (*Context Constraint*) A context constraint is a clause containing one or more context expression. It is satisfied if all its context expressions true. Otherwise it returns false.[2]



**Fig. 2.** Each S, R, O overlap, it has condition to execute

Context constraints are used to define conditional permissions. With respect to the terms defined above, a conditional permission is a permission associated with one or more context constraints. If the system grants subject for access, only context constraints evaluate to "true".

### 3.3 Subject

To allow for the specification of authorizations not only based on the user identity, but also on the user characteristics, each user is associated with one or more credential. A credential is a set of user attributes required for security purposes. Credentials[1][5] are assigned when a user is created, are updated automatically, according to the user's profile. A system distinguishes the correct user for accessing credential information. Definition7 is for the privilege specification process for user name, lifetime and privacy clearance level. In definition8, Privacy control is used compare P-CLR to P-CLS, and is based on sensitivity level [10]. Therefore subjects are assigned to role R,  $P\text{-CLR} \geq P\text{-CLS}$ .

**Definition 7.** (*User*) A User,  $U = [U^{User Id}, U^{LT}, U^{P-CLR}]$  is an active entity accessing a client, and satisfied a unique  $U^{User Id} \in USERS$ ,  $U^{LT} \in LT$ ,  $U^{P-CLR} \in SLEVEL$ .

**Definition 8.** (*User Authorization*)  $UA = [User^{Name}, Role, S\text{-Level}, CC]$  and assigned role using Privacy Control and Context Constraint.

### 3.4 Role

A role can represent a specific task competency, such as that of a physician or a pharmacist. A role is a named job function within the organization that describes the authority and responsibility conferred on a member of the role. Therefore access controls the invocation of an objects based on the role, the classification levels of the role and object, the time period when a role can invoke the object, and the object values under which a role is limited to invoke the object.

**Definition 9.** (*User Role*) A user role (*UR*)  $UR = [UR^{Name}, UR^{LT}, UR^{P-CLS}]$  uniquely represents a set of responsibilities, and satisfied a unique  $UR^{Name} \in ROLES$ ,  $UR^{LT} \in LT$  and  $UR^{P-CLS} \in SLEVEL$ .

**Definition 10.** (*User Role Authorization*)  $URA = [UR, O, LT, S\text{-Level}, CC]$  and executes the object of the role using Privacy Control and Context Constraint.

### 3.5 Object

The objects are data objects or resource objects represented by data in the computer system. Access control decisions can be made based on the various characteristics for objects.

Definition 11-12 defines objects and services. The service can approach an object by service class and object senior.

**Definition 11.** (*Object*) Each object  $O_{ij} = [O_{ij}^{Name}, O_{ij}^{LT}, O_{ij} + O_{ij}^{P-CLS}, O_{ij}^{CE}], O_{ij}(O_{ij}^{P-CLS})$  satisfied a unique  $O_{ij}^{Name} \in OBJECTS$ ,  $O_{ij}^{LT} \in O_{ij}^{P-CLS} \in SLEVEL$ ,  $O_{ij}^{CE} \in CONTEXT EXPRESSIONS$ .

**Definition 12.** (*Service*) Each Service  $S_i = [S_i^{Name}, S_i^{LT}, S_i + S_i^{P-CLS}, S_i^{CE}], S_i(S_i^{P-CLS})$  has name  $S_i^{Name}$ ,  $LT S_i^{LT} = [\min\{S_i^{LT}.st\}, \max\{S_i^{LT}.st\}]$   $S_i^{P-CLS} = \min\{S_i^{P-CLS} = \min S_i^{P-CLS} | i = l..m\}$ ,  $S_i^{CE} = \min\{S_i^{CE} | i = l..m\}$ .

Table 1. Privacy Control for Driver Role

Service (General Information)		Lifetime	Context Constraint	
Object	Name: Bob	[06.08.21, 07.05.30]	True	
	Address: 1408 Fox Run Trail Platte city MO	[06.09.30, 07.06.15]	True	
	Car Kind: Ford Mondeo	[06.01.20, 07.09.16]	True	
	Car Checked: 8 Reason: Engine trouble (5)	[06.02.01, 07.05.25]	True	
	Product Year: 2003.05	[06.01.05, 06.12.31]	True	
Service (Privacy Information)		Lifetime	Context Constraint	Alarms
Object (S-Level)	Driver Career (P-S): 3 year	[06.01.01, 07.11.30]	False	
	Driver Accident (P-S): 2	[06.06.01, 07.10.01]	False	
	Drunken Driving (P-TS): 1	[06.02.20, 07.12.31]	False	
	Car Title (P-S): 2003.06	[06.01.01, 07.12.31]	True	Checking

Table 1-2 is the example to solve access control between the driver and the garage based on definition 1-12 about previous problem. Role control decides the access availability of service by context constraint and lifetime.

## Assumption

1. All of the users assigns role by our mechanism.
2. The assigned driver role limits to fifth general information and four privacy information.
3. The garage is company to sign at certificate authority.

Bob be assigned the driver role of S-clearance by situation information among many the roles such as table 1. If the car is out of order in the road as previous problem, new mechanism not sends all information of the role like existing solution. The driver role separates general information and privacy information by proposed methods. General information limits the approach by role control, lifetime, and context constraint. And privacy information applies MAC concept to protect privacy besides them. As shown table 2, the garage obtains the information of the customer to request the service. But, the garage gets the information

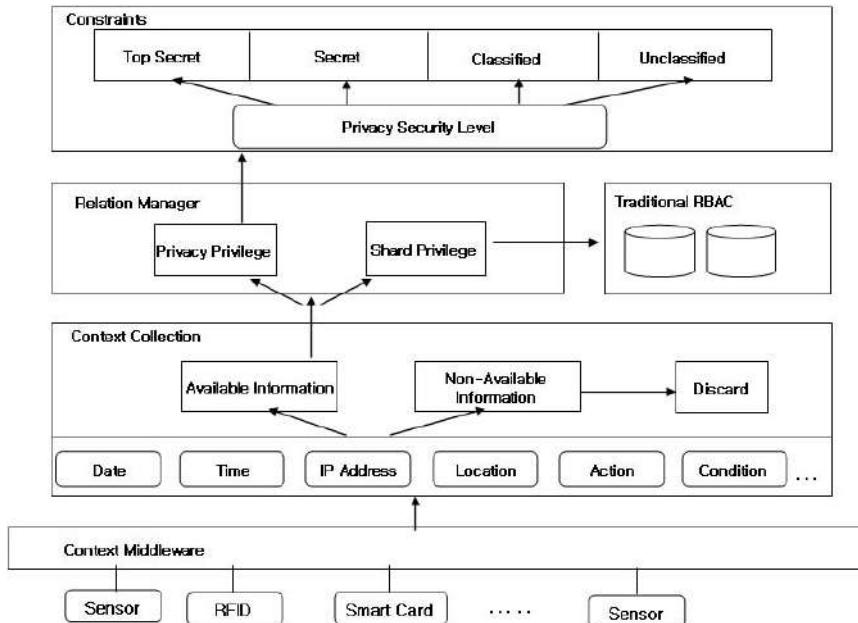
**Table 2.** Privacy Control for the garage

Service (General Information)	1. Name: Bob 2. Address: 1408 Fox Run Trail Plate city MO 3. Car Kind: Ford Mondeo 4. Car Check: 8 Reason: Engine trouble (5) 5. Product Year: 2005.05	1. Name: John 2. Address: 202 Monroe St Passaic, NJ 3. Car Kind: BMW X3 4. Car Check: 2 Reason: - 5. Product Year: 2005.08	1. Name: Robert 2. Car Kind: Audi A4 3. Car Check: 1 Reason: Tire change 4. Product Year: 2005.11
Service (Privacy Information)	1. Car Title (P-S): 2003.06	1. Driver Accident (P-S): 2 2. Car Title (P-S): 2005.11	

of different customer respectively by proposed mechanism. The first Bob case, the garage is the case to get the information of the car license to confirm whether they are a stolen vehicle with general information. The second John case, the garage is the case to get the information of the car license and driver accident career to confirm because it need of the state of car. The third Robert case, the garage obtained only fourth general information such as table 2. The reason is because role information became the violation by the restriction condition. Also if the privacy information is empty like table 2, we are the case which the garage did not request the privacy information or Robert rejects the request.

## 4 Secure Context RBAC/MAC Architecture

Sensitive private information must be protected from illegal access. In a ubiquitous computing environment it is essential to protect the privacy of users. The secure context-privacy architecture should deal with related access, privacy control, and context information (see Fig3). Context Middleware[3,4,18]



**Fig. 3.** Secure Context RBAC/MAC Architecture

provides user's location and situation information from sensor, RFID, and Smart card. For example, the current user situation(date, time, IP address, so on). In addition, Context Middleware includes managing functionality and filtering the potentially large amount of situation information that can be generated. The Context Collection checks the authority information of the user among the situation information. And it removes the remainder information. The Relation Manager provides privacy privileges and shared privileges using system access control policies. If a user requests privacy information in the current situation, the user accesses privacy control relating to the role of confidentiality and privacy protection. The context constraints can be applied to privacy control policies based on individual. But a shard privilege applies the traditional RBAC mechanism because it is a common role privilege.

## 5 Secure Context RBAC/MAC Policy

In the Context-Privacy access control environment, an access control request has three parameters ( S,O,P,C ) : subject(S), object(O), role permission(P), current context(C). Each subject has an authorized role set, consisting of all the roles that the subject has authorized for use. Permission[10,11,14] refers to the approval of a particular mode of access to one or more objects in the system. Objects are data resource objects represented by data in the computer system.

**Algorithm: Secure Context RBAC/MAC Algorithm**

**INPUT:** Query <Subject, Object, Permission, Context>

**OUTPUT:** Decision d {Accept, Deny, N/A}

Let Secure Privacy returned by function evaluate query <S, O, P, C>

For  $\forall u \in CU$  (Set of CREDNITALS types u:users r:roles)

// A set of roles can be activated for a user.

If  $\forall cu \rightarrow 2^R$  then  $(u, r \in C)$

Create set Role(URA=<UR, O, S-level, CC>), (User-Role assignment)

// Privacy and Shard permission decision

If  $\exists query \in Privacy\ Permission$  then

Else Create set Shard Permission (p) { $p \in Permission | (p,r) \in PA$ }

// S-Level(Privacy control)

// P-CLS(Role privacy security Level)

// P-CLS(Object privacy security Level)

If  $UR_{CLR} > O_{CLS}$  then

/\*LT/TC check UR's LT, the object's LT, and the TC of the URA must all overlap.\*/

If  $UR^{LT} \wedge O^{LT} \wedge TC \neq \emptyset$  then

/\*Permissions assigned to a user role when a given set of context roles is active.\*/

If  $PA \subseteq P \times R \times 2^{CR}$  then

Create set Privacy Permission (p) { $p \in Permission | (p,r) \in PA$ }

Else initialize

**Fig. 4.** Secure Context RBAC/MAC Architecture

Context is a subject's situation information. A request in the model comes from a certain user or subject S, with a set of associated roles.

For example if subjects request  $\langle S, O, P, C \rangle$  from the system, the system assigns a suitable role to the subject. And the system determines the authority whether the request is the individual through a situation information analysis of the user. It enforces access control if the request of the user is a shard authority. However, if a role refers to privacy privileges, an access control is handled by Fig4.

## 6 Conclusion

In this paper, we proposed a mechanism to develop context-aware access control model based on ubiquitous environment. The mechanism presented context-aware integration model based on both role-based and mandatory polices. We

are focused on solving the problem of securing privacy policies in ubiquitous computing. And defined how it to express access control polices.

In addition, we defined context RBAC/MAC that supports security policies that make use of context rules to control access to objects. The features of the model include context, privacy, and security mechanisms, and the ability to dynamically reconfigure access control polices to create different polices for ubiquitous environment.

As part of future work, we plan to extend our approach by distributed authorization framework underlying our current approach.

## References

1. Elisa Bertino, Ravi Sandhu. Database security-Concepts, Approaches, and Challenges. IEEE Transaction Vol.2, No.1 (2005) 2-19
2. Gustaf Neumann, Mark Strembeck. An Approach to Engineer and Enforce Context Constraints in an RBAC Environment, Proceeding of the eighth ACM symposium on Access control models and technologies. (2003)
3. Manuel Roman, Roy H. Campbell, and Klara Nahrstedt. Gaia OS: A middleware infrastructure to enable Active Spaces. IEEE Pervasive Computing, (2002) 74-83
4. Geetanjali Sampemane, Prasad Naldurg, and Roy H. Campbell. Access control for Active Spaces. In Proceedings of the Annual Computer Security Applications Conference (ACSAC) Las Vegas (2002)
5. N.R. Adam, V. Atluri, E. Bertino, and E. Ferrari. A Content-Based Authorization Model for Digital Libraries. IEEE Transactions on Knowledge and Data Engineering, (2002) 103-122
6. Hada, S. and Kudo, M.XML Access Control Language: Provisional Authorization for XML Document, October 2000, Tokyo Research Laboratory, IBM Research.
7. XACML and OASIS Security Services Technical Committee. eXtensible Access Control Markup Language (xacml) committee specification 2.0. 2005.
8. Gustavo H. M. B. Motta, Sergio S. Furui. A Contextual Role-Based Access Control Authorization Model for Electronic Patient Record. IEEE Transactions on Information Technology in Biomedicine Vol.7 NO.3 (2003) 202-207
9. X.Jiang, J. Hong and J.Landay. Approximate Information Flow: Socially Based Modeling of Privacy in Pervasive Computing. To be published in proceeding. Pervasive Computing, Springer-Verlag, Berlin (2002)
10. Charles E. Phillips, Stenen A. Demurjian. Security Assurance For an RBAC/MAC security Model, Proceeding of the IEEE, Workshop on Information Assurance N.Y(2003) 260-267
11. Matthew J. Moyer, Mustaque Ahamed. Generalized Role-Based Access Control, Distributed Computing Systems, Proceeding of the IEEE, 21st International Conference. (2001) 391-398
12. William Tolone, Gail-Joon Ahn,, and Tanusree Pai. Access Control in Collaborative Systems, ACM Computing Surveys (CSUR), Vol. 37 Issue 1. (2005)
13. Elisa Bertino, Barbara Catania, Elena Ferrari, and Palolo Perlasca. A System to Specify and Manage Multipolicy Access Control Models, Proceeding of the IEEE, Distributed Systems and Networks.(2002) 116-127
14. Michael J.Convington, Wende Long, Srividhya Srinivasan. Securing Context-Aware Applications Using Environment Roles, Proceeding of the sixth ACM symposium on Access control models and technologies. (2001)

15. R.S. Sandhu, E. J. Cynek, H. L. Fensteink, C.E. Youmank, Role-Based Access Control Model, IEEE Computer, Vol. 29, No.2, February (1996)
16. Jason Crampton. Specifying and Enforcing Constraints in Role-Based Access Control, Proceeding of the eighth ACM symposium on Access control models and technologies (2003)
17. Ahn, G. -J, And Sandhu, R. Role-based authorization constraints specification. ACM Transactions on Information and System Security Vol.3 issue4 (2000)
18. Corradi, A. Montanari, R. Tibaldi, D. Context-based access for pervasive service provisioning, Proceedings of the 28th Annual International Vol.1 (2004) 444-451

# Extending PostgreSQL to Support Distributed/Heterogeneous Query Processing

Rubao Lee<sup>1,2</sup> and Minghong Zhou<sup>1,2</sup>

<sup>1</sup> Research Centre for Grid and Service Computing, Institute of Computing Technology,  
Chinese Academy of Sciences

<sup>2</sup> Graduate University of Chinese Academy of Sciences,  
PO Box 2704, 100080 Beijing, China  
`{lirubao,zmh}@software.ict.ac.cn`

**Abstract.** The evolution from relational DBMS to data integration system brings new challenges to the design and implementation of query execution engine that must be extended to support queries over multiple distributed, heterogeneous, and autonomous data sources. In this paper, we introduce our work on extending PostgreSQL to support distributed query processing. Although PostgreSQL has no built-in distributed query processor, its function mechanism provides possibilities for us to integrate data of various data sources and execute distributed queries. We point out several limitations in PostgreSQL's query engine and present corresponding query execution techniques to improve performance of distributed query processing. Our experimental results show that the techniques can significantly reduce response times when running a workload consisting of TPC-H queries.

**Keywords:** Distributed Database, Query Processing, Performance.

## 1 Introduction

Modern data intensive applications need to integrate data from multiple distributed, heterogeneous, and autonomous data sources. To support such applications, traditional relational database management systems need to be extended to data integration systems that provide consistent data views on top of various data sources and support efficient distributed query processing to answer queries over the consistent data views [1][2].

The evolution from DBMS to data integration systems brings two basic challenges. The first challenge is how to make data integration using DBMS to be possible. Because data sources may have various differences in access interface and data storage, DBMS must provide a flexible and extensible architecture to enable various sources to be plugged into the system. This is similar to the problem in UNIX kernels that must provide support for various hardware devices using a consistent device driver interface. The second challenge is how to improve the performance of executing distributed queries considering the fundamental change from traditional disk access to distributed data access.

In data integration, all data are stored in remote data sources and can only be accessed through specific access interfaces provided by data sources. This is totally different from traditional DBMS that store all data in local storage devices like magnetic disks and access data through the file system interface provided by the underlying operating system. This essential difference makes it necessary for us to review existing designs and implementations of query execution engines in order to reduce initial delays and total response times when executing distributed queries. Several new factors must be considered, such as network latency, network bandwidth, and capabilities of data sources.

This paper presents our work on extending PostgreSQL to a data integration system. PostgreSQL is a traditional relational DBMS that has no built-in support to distributed query processing. However, the function mechanism [3] of PostgreSQL provides a way to extend the capability of PostgreSQL backend, which forms the foundation on which users can access and integrate data of various data sources inside PostgreSQL. This makes data integration to be possible, however, the performance of executing distributed queries through the function interface in PostgreSQL cannot be improved if we do not modify the query execution engine. Because a function is a “black box” for the engine that can only invoke the function through a specific interface, the interval execution of the function cannot be optimized by the engine.

We highlight two features of our extension in this paper. First, we provide a well-defined interface of data source wrappers, which enables various data sources to be plugged into the system. Second, we create and implement several key query execution techniques in the engine of PostgreSQL so that the performance of executing distributed queries can be significantly improved. To our knowledge, we are the first to add distributed query processing in PostgreSQL, although similar extensions to commercial DBMS products have been proposed [4][5].

This paper is organized as follows. Section 2 presents an overview of adding distributed query processing in PostgreSQL. In section 3, we discuss the performance issues of executing distributed queries in PostgreSQL and introduce corresponding query execution techniques. Section 4 describes various experimental results. Related work is introduced in section 5. We conclude this paper and introduce our future work in section 6.

## 2 Adding Distributed Query Processing in PostgreSQL

PostgreSQL provides function mechanism which allows developers to implement various extensions to the backend. We can utilize the function mechanism to add distributed query processing in PostgreSQL. For example, we can create a function that returns all tuples of a table located in a remote Oracle database and create a view defined on the results of invoking that function. The function concept is a way of achieving “resource virtualization” that creates a mapping form the view object in PostgreSQL to the physical table in the data source, which is the core of data integration.

## 2.1 Data Source Wrapper

A data source wrapper is a software component between PostgreSQL and a data source, which is like a device driver between the UNIX kernel and a hardware device. The wrapper component hides the differences in access interface and data storage of various data sources and provides an abstraction to enable sources to be plugged into the system. From the point of view of query engine, a wrapper is a set of functions. We define the interface of each function and allow for various implementations using any programming language supported by PostgreSQL.

There are two kinds of data source wrappers. The first is for data sources which can execute standard SQL statements, such as a DBMS. Such a wrapper must implement a function that takes a SQL statement as one of arguments and returns result tuples of executing the input SQL statement in the data source. The second is for data sources which have no support of executing SQL, such as a web page or a web service. The corresponding function in such a wrapper takes an object name as the argument and returns all tuples converted from the contents of that object. For any kind, it is the wrapper that is responsible for transforming specific data types in data sources to data types supported by PostgreSQL.

## 2.2 Extended DDL Statements

We provide several extended DDL statements for data integration in PostgreSQL. We extend the system catalog to store various metadata information, such as properties of data sources, mappings between local views and remote objects, etc. When processing distributed queries, the engine needs to utilize the metadata information.

We describe three important DDL statements. The first statement is used to register a data source wrapper. It must provide the locations of files implementing the wrapper functions and must tell PostgreSQL whether the wrapper has capability of executing SQL or not so that the query engine can determine whether it can push part of a query down to the data source or not. The DDL statement is like:

```
CREATE DATASOURCE WRAPPER wrapper_name (INSTALL
SCRIPT : 'file_path', EXEC FILE : 'file_path',
CAPABILITY : SQL_SUPPORT | NO_SQL_SUPPORT)
```

The second statement is used to register a data source. It must specify which wrapper needs to be used to wrap the data source and must provide the “Access Path” of the data source which is interpreted and utilized by the wrapper. For example, the “Access Path” may be a DSN name for an ODBC data source. The DDL statement is like:

```
CREATE DATASOURCE datasource_name (ACCESS PATH : 'a
string', WRAPPER: wrapper_name)
```

The third statement is used to register an object located in a data source. It must provide the data source name and the object’s unique name in the data source. After executing the statement, a view is created in PostgreSQL which is mapped to the remote object. The DDL statement is like:

```
CREATE RESOURCE resource_name (DATASOURCE :
datasource_name, LOCAL ID : 'object's ID')
```

We take an example to illustrate the above DDL statements. Assume that a table *t1* is located in an Oracle DBMS with IP address “10.0.1.1” and a table *t2* is provided by a web service with URL “[http://10.0.1.2:8080/multi\\_tables](http://10.0.1.2:8080/multi_tables)”. The following DDL statements can be used to register *t1* and *t2* into PostgreSQL.

```
(1): CREATE DATASOURCE WRAPPER wrapper_for_oracle (
INSTALL SCRIPT: '/tmp/oracle.sql', EXEC FILE
'/tmp/oracle.bin', CAPABILITY: SQL_SUPPORT);

(2): CREATE DATASOURCE ds_oracle (ACCESS PATH:
'10.0.1.1/db/username:passwd', WRAPPER:
wrapper_for_oracle);

(3): CREATE RESOURCE ds1.t1 (DATASOURCE: ds_oracle,
LOCAL ID: 't1');

(4): CREATE DATASOURCE WRAPPER wrapper_for_ws (INSTALL
SCRIPT: '/tmp/ws.sql', EXEC FILE '/tmp/ws.bin',
CAPABILITY: NO_SQL_SUPPORT);

(5): CREATE DATASOURCE ds_ws (ACCESS PATH:
'http://10.0.1.2:8080/multi_tables', WRAPPER:
wrapper_for_ws);

(6): CREATE RESOURCE ds2.t2 (DATASOURCE: ds_ws, LOCAL
ID: 't2');
```

After executing these DDL statements in PostgreSQL, two views *ds1.t1* and *ds2.t2* are created in the database and user can submit queries over the views, such as: “*select \* from ds1.t1 natural join ds2.t2*”. So far, data integration is realized in PostgreSQL. In the next section, we will discuss the performance issues.

### 3 Query Execution Techniques

The function mechanism makes distributed query processing in PostgreSQL to be possible, however, because PostgreSQL is not designed for processing distributed queries in nature, some limitations in the query engine exist, so that performance of distributed query processing is limited. In this section, we first point out these limitations and then present corresponding query execution techniques.

#### 3.1 Problems of Processing Distributed Query in PostgreSQL

The query execution engine in PostgreSQL is not optimized for execution of functions because the implementation of a function is a “black box” for the engine. To reduce initial delays and total response times of processing distributed queries, we have to overcome several key limitations existing in the query engine of PostgreSQL.

The first limitation is the materialization policy in implementing FunctionScan operator. This limitation makes it impossible for clients to obtain initial results rapidly when issuing a distributed query. When the Next function of FunctionScan operator is

first invoked, the operator will first store all result tuples returned by invoking the underlying function into a temporal buffer and then return the first tuple in the buffer to the parent operator. If the underlying function is used to fetch all tuples from a remote table in a data source, then in order to get the first result tuple the parent operator will have to wait for a long time until all tuples have been received by the FunctionScan operator. This behavior is not acceptable for applications especially involving Top-K queries.

The second limitation is the fact that PostgreSQL lacks query shipping mechanism [6] when we add distributed query processing in it by utilizing its function mechanism. For example, even though a FunctionScan operator has a filter which indicates which tuples are needed by the parent operator in the query plan tree, the FunctionScan operator still has to fetch all tuples of the corresponding object in the remote data source without excluding those tuples that cannot pass the filter. Obviously, pushing the filter down to the data source can reduce the number of result tuples transferred over the network. To implement query shipping based on the FunctionScan operator, the query engine must dynamically adjust arguments of the underlying function.

The third limitation is the single-threaded implementation of the query engine in PostgreSQL. For a distributed query involving multiple remote data sources, the single-threaded query engine can only interact with all the data sources using a sequential and synchronized way. In this way, once a data source cannot return next tuples immediately, the whole query engine process will have to be blocked. Ideally, when the query engine process is blocked by a data source, it still can obtain data from other data sources by thread scheduling. However, the fact that codes of the query engine are not thread-safe makes it difficult to achieve this goal inside the query engine.

In the following subsections, we present three query execution techniques to overcome these limitations.

### 3.2 Pipelined Data Fetch

To reduce the initial delay of executing a distributed query, a pipelined data path for transferring data from the data source to the query engine is required. To archive this goal, we re-implement the FunctionScan operator using a non-blocking policy. Whenever the Next function of the FunctionScan operator is invoked, the operator will fetch next tuple from the underlying wrapper. Therefore, the parent operator in the query plan tree can rapidly obtain tuples from the FunctionScan operator without being blocked until the FunctionScan operator receives all result tuples from the wrapper.

Moreover, to achieve such a pipelined data path, the underlying wrapper must also be implemented using a non-blocking policy. However, non-blocking execution of a wrapper may be limited by the data source's capability. For data sources with capability of executing SQL statements, we can implement a pipelined wrapper by holding a cursor to fetch more tuples on demand.

For complex queries involving join across multiple remote relations, reducing initial delays needs efficient algorithms, such as XJoin [7]. To implement such algorithms in the query engine, the pipelined execution in FunctionScan and wrappers must be provided as the prerequisite.

### 3.3 Query Shipping

Query shipping is to push some computations in the query plan down to data sources instead of executing them by the query engine of PostgreSQL. Query shipping is only applied to data sources which can execute SQL statements. We implement query shipping by adjusting arguments of the function dynamically in the execution of the FunctionScan operator.

We take an example to illustrate this dynamic adjusting. Assume that “*rt\_t1*” is a view defined on the result of a function `remote_execute` with the argument “*select \* from t1*”. When executing a query over “*rt\_t1*”, the argument string of the function can be dynamically replaced. For example, when executing the query “*select \* from rt\_t1 where field\_1 > 100*”, the corresponding FunctionScan operator can dynamically replace the argument of the function to “*select \* from t1 where field\_1 > 100*” so that execution of the filter can be pushed down to the data source.

Currently, our query shipping implementation supports three kinds of operations including selection, projection, and sorting. By pushing selection and projection down to the data source, the amount of data transferred over the network, i.e. the number of tuples or columns, can be reduced so that the total response time of query execution can be reduced. Unlike this, the benefit of pushing sorting down is mainly the reduced initial delays, as illustrated by experimental results shown later. The default Sort operator in PostgreSQL is a blocking operator that cannot output the first tuple before it fetched all tuples from its child operator and then sorted them. However, after pushing sorting down the data source, the Sort operator only needs to transmit tuples fetched from its child operator, since they have already been sorted by the data source. Especially, if the data source has additional support for sorting, such as index or materialized sorting results, the initial delay can be reduced further. However, whether and how well the total response time can be reduced by pushing down sorting is dependent on the performance of executing sorting in data source.

### 3.4 Start-Fetch

PostgreSQL employs a traditional “one connection, one process” model to execute each query. In this model, execution of FunctionScan operator is within the same process of the query engine. If FunctionScan is blocked for some reasons caused by network transfer or data source response, the whole process will have to wait until FunctionScan obtains data. During the period of waiting, nothing can be done even though the query engine can receive data from other data sources.

The goal of Start-Fetch is to utilize intra-query parallelism to hide unnecessary network latency on the basis of single-threaded query engine. The main idea behind

Start-Fetch is to decouple the wrapper execution from the query engine process to improve parallelism between them. We implement the wrapper as an independent process and employ a shared-memory mechanism to connect the wrapper and the query engine.

In nature, Start-Fetch is a way that the query engine process interacts with a wrapper process based on the iterator execution model [8]. When the Open function of FunctionScan operator is invoked, the query engine sends request to the wrapper and the wrapper must immediately return a “ticket” to the query engine. Then, the wrapper needs to send the request to the data source and receive results independently in its own process. This is the “Start” step in Start-Fetch. When the Next function of FunctionScan operator is invoked, the query engine asks for next tuple from the wrapper using the ticket obtained in the “Start” step. This is the “Fetch” step in Start-Fetch. The decoupling policy makes parallelism between the query engine and multiple wrappers to be possible. Start-Fetch provides two benefits. First, for a query involving multiple data sources, initial delay of each wrapper for waiting results from data source will not be accumulated because all requests can be nearly simultaneously sent to the data sources. Second, independent wrapper process can prefetch more tuples from data sources while the query engine is consuming old tuples.

## 4 Experimental Results

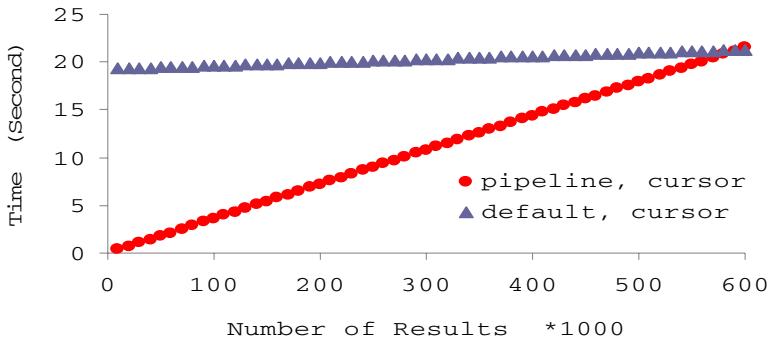
In this section, we present the experimentation with our extended PostgreSQL. Considering no common data integration query benchmark, we use TPC-H [9] (scale factor 0.1, 100MB) as the basis of our experiments. We use two data sources. The first one is a Microsoft SQL Server running on a Windows 2003 Server box with a 2.4GHz Pentium IV CPU and 1.5 GB of memory. The second one is a PostgreSQL 8.1 server running on a machine as same as the machine of data source 1. We implement two wrappers. For the SQL Server, we use unixODBC as the client library. And for the PostgreSQL, we use libpq. Our extended PostgreSQL is running on a machine with a 2.8GHz Pentium IV CPU, 768MB of memory, and a FreeBSD 5.4 as the operating system. All machines are connected using a 100Mbit/sec Ethernet.

We load the 100MB dataset into each data source. We register each table of data source 1 into our extended PostgreSQL as a homonymic view under the schema *ds1* and data source 2 under the schema *ds2*. For example, the view “*ds2.lineitem*” is mapped to the table “*lineitem*” in the data source 2.

### 4.1 Pipelined Data Fetch

In this experiment, we examine how well the pipelined FunctionScan and wrapper can reduce the initial delay when executing a query returning many tuples. The query is “*select \* from ds2.lineitem*”. The results are shown in Figure 1, which present the

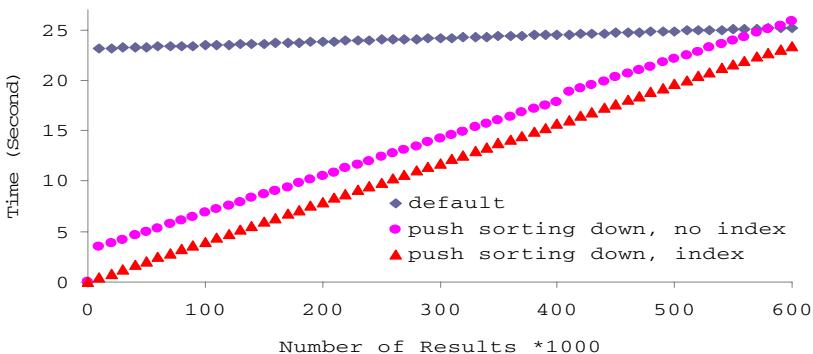
comparison of elapsed times for obtaining different numbers of results. Using a pipelined data fetch can reduce the initial delay to a very low value without sacrificing the total response time.



**Fig. 1.** Pipelined data fetch in FunctionScan can significantly reduce the initial delay without sacrificing the total response time. The number of result tuples is 600572.

## 4.2 Query Shipping (Sorting)

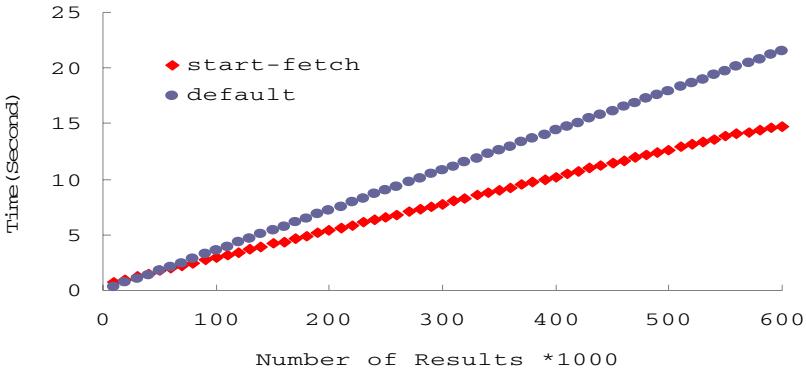
In the next experiment, we examine how the query execution can benefit from pushing sorting down. In the experiment, the query is “*select \* from ds2.lineitem order by l\_orderkey*”. Figure 2 shows the results. By pushing sorting down to the data source, the initial delay can be significantly reduced, especially when the data source has index on the column *l\_orderkey* of table *lineitem* to accelerate sorting, although the total response time is not reduced too much.



**Fig. 2.** By pushing sorting down, the initial delay can be significantly reduced especially when the data source has index to help sorting. The number of result tuples is 600572.

### 4.3 Start-Fetch

To examine Start-Fetch, we do two experiments. In the first experiment, we use the query “*select \*from ds2.lineitem*” to examine how prefetching in the wrapper process can accelerate the query processing. Figure 3 shows that by prefetching next tuples proactively, the total response time can be reduced by about 30%.

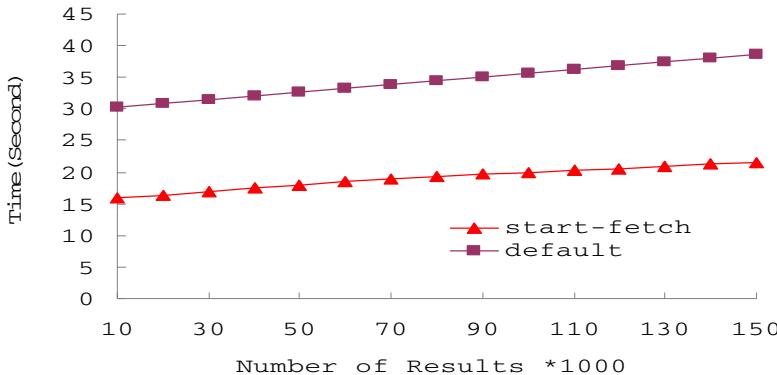


**Fig. 3.** By using Start-Fetch, the wrapper can prefetch next tuples while the query engine is consuming old tuples so that the total response time can be reduced

In the second experiment, we examine how Start-Fetch can hide unnecessary initial delays when executing a query involving multiple data sources. We create a view “*orders\_view*” in each data source and register them into our extended PostgreSQL. The view definition is:

```
create view orders_view as select * from orders where
o_orderkey in (select l1.l_orderkey from lineitem l1,
lineitem l2 where l1.l_orderkey = l2.l_orderkey and
l1.l_suppkey = l2.l_suppkey and l1.l_partkey =
l2.l_partkey)
```

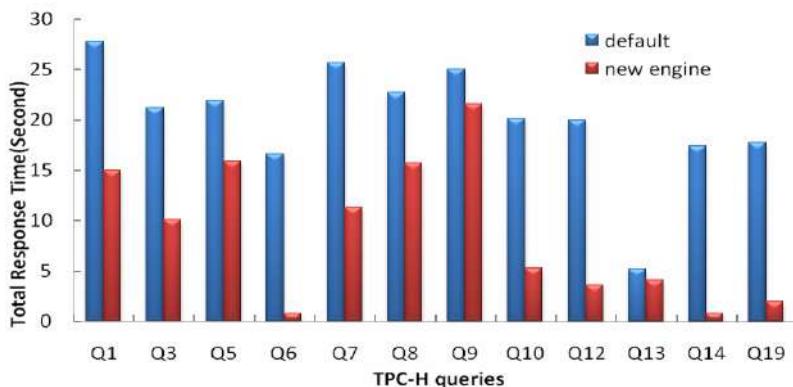
Although the view “*orders\_view*” contains as same tuples as the table *orders*, there will be a long initial delay to obtain tuples from it. We execute a query “*select \* from ds1.orders\_view o1, ds2.orders\_view o2 where o1.o\_orderkey = o2.o\_orderkey*”, and force the optimizer to choose sort-merge join in the query plan, so that each data source will be requested to execute the query “*select \* from orders\_view order by o\_orderkey*”. Figure 4 shows that the initial delay and the total response time can be reduced to 50% if Start-Fetch is enabled. This is because the two data sources can receive requests from the engine almost simultaneously and then process their own request independently. By the default execution, however, two requests will be sequentially sent to data sources so that their initial delays will be accumulated.



**Fig. 4.** Start-Fetch exploits intra-query parallelism to reduce the initial delay and the total response time when executing the query involving multiple sources

#### 4.4 Running TPC-H Queries

In the next experiment, we test how the combination of all three query execution techniques can improve the performance when running TPC-H queries. We use TPC-H queries #1, #3, #5, #6, #7, #8, #9, #10, #12, #13, #14, and #19. In these queries, the relation for lineitem is the one located in data source 1, and all other relations are located in data source 2. Figure 5 shows that our query execution techniques can significantly reduce total response time of executing each query.



**Fig. 5.** Total response times of executing TPC-H queries

## 5 Related Work

Several DBMS vendors extend their own DBMS products to support distributed query processing. In [5], IBM introduces the distributed extension of DB2 on the basis of

the Garlic prototype. In [4], Microsoft introduces how to employ the OLE DB data access interface to support distributed/heterogeneous query processing in SQL Server. Compared with these commercial products, our extension to PostgreSQL is similar in system architecture and wrapper usage. However, currently we do not provide a distributed query optimizer as mentioned in these papers and in [10], which is part of our future work. Nevertheless, the Start-Fetch technique in our extended PostgreSQL is unique.

Many papers are related to distributed query processing. An overview of distributed query processing can be found in [1]. In [6], the performance comparison of data shipping, query shipping, and hybrid shipping in client-server query processing is presented. Adaptive query execution techniques in data integration can be found in [11][12]. The XJoin algorithm is introduced in [7], which is an adaptive scheduling-based pipelined hash join algorithm, and whose multi-join version can be found in [13]. We are studying XJoin and considering implementing it in our extended PostgreSQL.

Several papers cover extending PostgreSQL to support new applications. Telegraph [14] is a dataflow processing system based on PostgreSQL. In [15], the authors introduce integrating active databases with publish/subscribe using PostgreSQL and Hermes as the experimental context.

## 6 Conclusions and Future Work

The evolution from DBMS to data integration systems brings new challenges to the design and implementation of query execution engine that must be extended to support queries over multiple distributed, heterogeneous, and autonomous data sources. In this paper, we introduce our work on extending PostgreSQL to support distributed query processing. Our distributed extension to PostgreSQL is based on a well-defined wrapper interface which employs the function mechanism to achieve the goal of integrating data of various data source. However, due to the fact that PostgreSQL is not designed for supporting distributed query processing in nature, in order to improve the performance of executing distributed queries, we have to overcome several limitations in the query execution engine, which include the non-pipelined FunctionScan operator, the lack of query shipping mechanism and synchronized query execution. We present three corresponding query execution techniques to reduce initial delays and total response times when executing distributed queries. Our experimental results show that these techniques can significantly improve performance of our extended PostgreSQL when running a workload consisting of TPC-H queries.

In the near future, we plan to (a) implement a distributed query optimizer in PostgreSQL, (b) to implement an XJoin-like pipelined hash join, and (c) to study how to accelerate executing distributed queries containing subqueries such as the TPC-H query #4.

**Acknowledgments.** This work is supported in part by National Natural Science Foundation of China (Grant No. 90412010 and No.60403023) and the China National 973 Program (No. 2005CB321807).

## References

1. D. Kossmann. "The State of the Art in Distributed Query Processing." ACM Computing Surveys, 32(4), December 2000, pp.422-469.
2. Z. G. Ives. "Efficient Query Processing for Data Integration." PhD thesis, University of Washington, August 2002.
3. M. Stonebraker and G. Kemitz. "The POSTGRES Next Generation Database Management System". In Communications of ACM, 34(10),1991, pp.78-92.
4. J.A. Blakeley, C. Cunningham, N. Ellis, B. Rathakrishnan, and M.C. Wu. "Distributed/Heterogeneous Query Processing in Microsoft SQL Server." In *Proc. ICDE*, 2005
5. V. Josifovski, P. Schwarz, L. M. Hass, and E. Lin. "Garlic: a New Flavor of Federated Query Processing for DB2". In *Proc. SIGMOD*,2002
6. M. J. Franklin, B. T. Jonsson, and D. Kossmann. "Performance tradeoffs for Client-Server Query Processing." In *Proc. SIGMOD*,1996.
7. T. Urhan and M. J. Franklin. "XJoin: A reactively-scheduled pipelined join operator." IEEE Data Engineering Bulletin, 23(2), June 2000, pp.27-33.
8. G. Graefe. "Query evaluation techniques for large databases." ACM Computing Surveys 25(2), June 1993, pp.73–170.
9. Transaction Processing Performance Council: <http://www.tpc.org/tpch/default.asp>
10. L. M. Hass, D. Kossmann, E. L. Wimmers, and J. Yang. "Optimizing queries across diverse data sources." In *Proc. VLDB*,1997.
11. Z. G. Ives, D. Florescu, M. T. Friedman, A. Y. Levy, and D. S. Weld. "An adaptive query execution system for data integration." In *Proc. SIGMOD*,1999.
12. Z. G. Ives, A. Y. Halevy ,and D. S. Weld. "Adapting to Source Properties in Processing Data Integration Queries" In *Proc. SIGMOD*, 2004
13. S. Viglas, J. Naughton, and J. Burger. "Maximizing the output rate of multi-join queries over streaming information sources." In *Proc. VLDB*, 2003
14. S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein,W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World." In *Proc. CIDR*, 2003
15. L. Vargas, J. Bacon, and K. Moody. "Integrating Databases with Publish/Subscribe." In *Proc. ICDCSW*,2005.

# Geo-WDBMS: An Improved DBMS with the Function of Watermarking Geographical Data

Min Huang<sup>1</sup>, Xiang Zhou<sup>2</sup>, Jiaheng Cao<sup>1</sup>, and Zhiyong Peng<sup>1,2</sup>

<sup>1</sup> Computer School, Wuhan University, wuhan, 430072, China

<sup>2</sup> State Key Lab of Software Engineering, Wuhan University, wuhan, 430072, China  
hm172213@hotmail.com

**Abstract.** This paper focuses on the issue of geographical data's copyrights protection. A Geo-WDBMS has been built by embedding the watermarking functions into the inner code of the open source DBMS PostgreSQL. And its core watermarking mechanism is to insert and detect mark bits in the coordinates of the vertices in geographical objects using the methods of classifying and twice majority-voting. Further more, error correcting mechanism is used to enhance the resilience of the system and blind watermark is realized. Experiments on watermarking digital maps showed that the marked maps are inconspicuous and robust to various attacks.

**Keywords:** copyrights protection, watermarking databases, geographical data.

## 1 Introduction

### 1.1 Background

Nowadays, pirates and unlimited duplicates of digital products have severely violated the owners' rights and interests, so the copyrights protection of digital products is getting more and more attentions. Database, as a kind of digital product, its copyrights protection is a hot issue in recent database research [1, 2, and 3], which is different from various studies of DBMS (such as security model, access control, and etc.). Digital watermark technology provides an effective solution for the problem which can confirm the data's ownership or verify the originality of digital contents by inserting perceptive or imperceptible information into digital products. With regard to relational databases, some intentional small errors in the data construct imperceptible watermark information.

Gathering data accounts for more than 80% of the cost of any GIS project. The reasons for the high cost are: on the one hand, many GIS enterprises are in a heavily need of geographical data; on the other hand, unlimited copies threatened the owners' benefits from geographical database, which made the exchanges of Geo-data very difficult. Outside the GIS community this problem has been known for a long time using watermarking technology. 2D vector and point datasets have received less attention from the research community; however, 3D meshes have been considered by the CAD community and a handful of techniques are available for that case [4]. So how

to protect the ownership of 2D Geo-data is our research point. The existent resolution mainly watermarks the vector map as graphics regardless of database aspects [5]. However, effectively combining watermarking technology and database technology is a trend in non-multimedia data's copyrights protection.

Watermarking geographical database provides a good resolution for this problem. In this paper, a Geo-WDBMS is built to protect the Geo-data's copyrights and it is implemented through embedding watermarking mechanism into the inner code of PostgreSQL with the support of PostGIS. PostgreSQL is an open source object-relational DBMS, and PostGIS adds support for geographic objects to the PostgreSQL. In effect, PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems (GIS), much like ESRI's SDE or Oracle's Spatial extension.

## 1.2 Related Works

Recent years, lots of researches focused on multimedia watermark technology (image, audio, video and etc.) and its theoretic system of evaluating the watermark methods has been set up. However, the study of watermarking database for copyrights protection just came forth in 2002 which was proposed by R.Agrawal in the 28<sup>th</sup> VLDB conference. Thus the solution to this ongoing issue is not full-fledged which can not be theoretically and formally demonstrated and evaluated.

Paper [6] proposed bit-resetting method which can mark the numeric attributes in relational databases. The basic idea is to reset the selected bit of a specific attribute value and to validate whether the database contains watermark through the threshold. But this method just embeds random and meaningless bit-flow, so it can only determine whether the watermark exists but gives no information of what the watermark is. What is more, when the attacker changes the schema of the relational table (e.g. cutting an attribute or simply re-sorting the attributes), it is almost impossible to detect the correct watermark.

Paper [7, 8] gives a distribution preserving method which inserts a “virtual” mark through adjusting the distribution of data in each subset. However, to adjust the data's distribution is time-consuming and the method is not desirable in withstanding distortion attack.

In the area of protecting the Geo-data's copyrights, most researches deal with digital map as still image and use image watermarking technology, while mark each copy of the map image is a laborious task. What is more, watermark may impact the topology of geographical objects in the map, and the customers can not analyze the data in the map image either. However, many applications need the data in the databases but not only the images.

Paper [10] puts forward the idea of watermarking the geographical databases with a secret watermarking method which does not accord with the request of public watermarking algorithms.

Paper [11] inserts watermark by overlap or interpolation of Geo-data, which tries to maintain the data's accuracy at the cost of increasing the quantity of points. However, the paper does not describe the method clearly and the interpolation method is CPU-consuming which is hard to meet the need of real-time applications.

Paper [12] proposes a digital watermarking algorithm for vector digital maps. A watermark bit is embedded by displacing an average of coordinates of a set of vertices that lies in a rectangular area created by adaptively subdividing the map. However, it needs original map to detect watermark and does not take the database watermark technology into account. We aim at blind watermark detection.

### 1.3 Our Contribution

The major contributions of this paper can be described in 3 aspects:

(1) It is novel to build a DBMS with the function of watermarking relational tables. To deal with huge amount of relational data, it is more efficient by using a WDBMS than using a watermarking API package.

(2) We proposed a watermark framework with error correction mechanism (WFEC), which pays much attention to correcting the errors in the detected watermarks by using BCH (Bose-Chaudhuri-Hocquenghem) coding method [9].

(3) The WFEC watermarking mechanism is embedded into the inner code of PostgreSQL and is capable of watermarking all types of geographical data (e.g. point, line, polygon and etc.) with the support of PostGIS. And the detection of watermark does not need original Geo-data compared with other related works which need original data to recover watermarks.

We propose WFEC method which inserts marks into the geographical databases using public watermarking algorithms. Thus mark bits are hidden in the specific objects and once the database is watermarked, all the digital maps based on it contain the marks. Twice majority-voting method makes up for some small mistakes in the detected watermark-bit caused by all kinds of attacks, so the final detected watermark won't be affected badly. BCH error correcting mechanism will enhance the success rate of watermark detecting, which leads to a better performance.

The rest of the paper is organized as follows. Section 2 gives the framework of the Geo-WDBMS and introduces general watermarking mechanism. Section 3 discusses watermarking algorithms. Section 4 provides the implementation of the mechanism in PostgreSQL and experiment evaluation. Section 5 makes a conclusion and points out the future work.

## 2 Framework of Geo-WDBMS

The Geo-WDBMS is built based on the PostgreSQL with the support of PostGIS and the function of watermarking is realized according to the following framework represented by Fig. 1.:

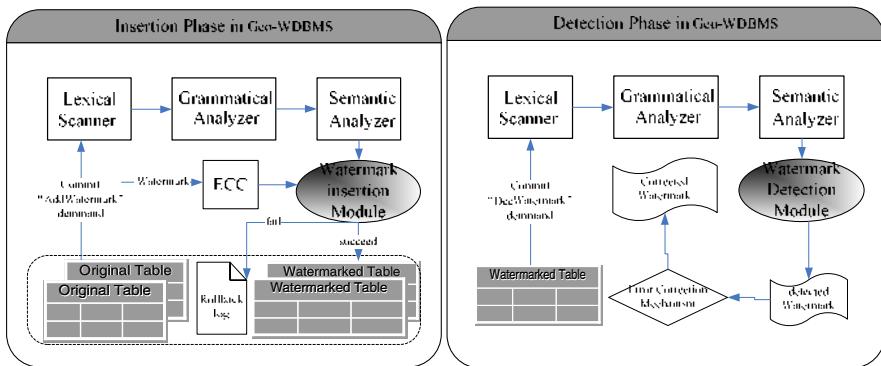
During the watermark insertion phase, when a geographical table is ready for watermarking, the Geo-WDBMS works as follows:

- (1) Receive the command “AddWatermark” and recognize the command;
- (2) Append the error correction code (ECC) to the watermark information to be embedded;
- (3) Execute the watermark insertion algorithm to watermark the table with the generated watermark, a provided secret key, and a set of watermark parameters;

(4) If the “AddWatermark” operation succeeds, write the marked data into the disk, otherwise rollbacks the operation and commits a rollback log.

During the watermark detection phase, for a geographical table which is suspected to be a pirated copy, the system works as follows:

- (1) Receive the command “DecWatermark” and recognize the command;
- (2) Execute the watermark detection algorithm to recover watermark information from the suspected copy;
- (3) Run error correction mechanism to correct the errors in the detected watermark;
- (4) Compare the corrected watermark with the original one to determine whether or not the relational table is piratical.



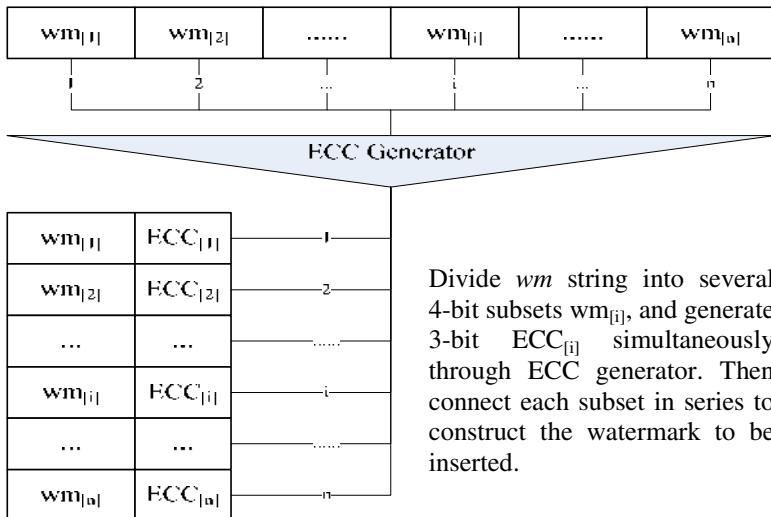
**Fig. 1.** Framework of the Geo-WDBMS

## 2.1 Error Correction Mechanism

Verifying-bits addition and error correction are used to recover from minor mistakes in the detected watermarks to improve resilience. Because any watermarked data will be faced with various attacks with a high probability, and an embedded watermark is very likely to be altered when attacks take place. Therefore, the introduction of error correction mechanism will make the watermark more robust against malicious attacks.

There are many error correcting codes applied in the digital watermarking systems to protect the embedded information against noises, such as BCH (Bose-Chaudhuri-Hocquenghem) codes [13], RS (Reed-Solomon) codes [14] and Turbo codes [15]. They are all widely used in communication realm, but BCH and RS codes are most common ones. BCH codes were demonstrated as good correcting codes and can correct errors up to approximately 25% of the total number of digits achieving Shannon limit performance. And it outperforms RS codes for the lower decoding complexity. As we all know, BCH uses binary coding method to encode messages, therefore, no evaluation process will be conducted once the error bit is located.

We use BCH (7, 4) code here, that is, the length of code word is 7 bits, the length of information code is 4 bits and it can correct one bit error.



**Fig. 2.** Pretreatment of watermarking information

Fig. 2 gives the pretreatment of watermarking information. When the user gives the watermark information  $wm$ , ECC generator produces the corresponding verifying-bits, that is:

$$\text{Verifying-bits} = \text{ECC}(wm);$$

Then Verifying-bits are appended to the original watermark, that is:

$$\text{Watermark to be inserted} = \text{Original watermark} + \text{Verifying-bits}.$$

### 3 Watermarking Algorithms of Geo-WDBMS

Based on the framework of Geo-DBMS in section 2, we proposed an amplified bit-resetting watermarking algorithm specified for Geo-data with error correction mechanism.

#### 3.1 Scheme of Watermarking Geo-data

Vector maps are stored in PostgreSQL as relational tables, and each real object in the map is stored as a tuple in the relational table. The content to represent the spatial information is stored in a special attribute, namely geometry attribute. The main members in the structure of GEOMETRY in PostGIS are defined as follows:

```
typedef struct
{
 int32 type;
 int32 nobjs;
 int32 objType[1];

} GEOMETRY;
```

An object may have several sub-objects, so in the GEOMETRY structure: ‘nobjs’ is the number of sub-objects, and ‘objType[1]’ is a length-variable array. ‘objType’ has 3 values: Point, Line and Polygon, which respectively refers to POINTTYPE, LINETYPE and POLYGONTYPE.

The basic idea of watermarking Geo-data is to insert marks into the coordinates of vertices in a map. There are three kinds of basic object types (point, line, and polygon) in GEOMETRY, and they construct complicated object (multipoint, multiline, and multipolygon). First, each sub-object in a tuple will be uniquely identified, that is, the identifying granularity is the basic object type; then, they are dealt with according to their distinct characteristics respectively, which will be discussed in detail in section 3.2.

For simplifying the problem, we describe the schema of relational table as  $R(MSA, attr_1, \dots, attr_n)$ :  $MSA$  includes at least one attribute and each will be noted as  $A_i$ . Assume  $attr_1, \dots, attr_n$  are all watermark candidate attributes. There are  $\Omega$  tuples in table  $R$ . Any tuple in  $R$  is represented by  $t$  and the values of attri in tuple  $t$  are noted as  $t.attri$ .

### 3.2 Watermark Insertion

To watermark a relational table  $R$  ( $MSA, attr_1, \dots, attr_n$ ), the parameters used in watermark inserting are defined as follows:

**Table 1.** Notation

$\kappa$	secret key
$attri$	candidate attribute
$mark\_info$	original watermark
$1/\lambda$	Fraction of watermarked tuples to total tuples
$\zeta$	Number of the least significant bits in a value
bounds	data usability constraints

- ① Preprocess mark\_info: in regard to the properties of BCH (7, 4) code, we divide the original watermark bit flow into 4-bit sub-watermark noted as  $sub-watermark_{[i]}$  which will be the input of ECC generator to produce verifying-bits. Thus we get the final form of the watermark information =  $\{sub-watermark_{[1]} \circ verifying-bits_{[1]} \dots sub-watermark_{[1]} \circ verifying-bits_{[1]} \dots sub-watermark_{[n]} \circ verifying-bits_{[n]}\}$  which is shown in Fig. 2. Then repeat watermark information for  $repeatnum$  times.
- ② Calculate the id of each object: if an object contains several sub-objects, calculate the id of each sub-object.
- ③ Sort all the sub-objects by their ids and divide them into several equal-sized subsets, and the number of subsets equals to the bit length of watermark produced in step ①. The first object’s id of each subset is kept in the array  $subset\_boundaries[]$ .
- ④ Each subset will be marked with one bit of watermark repeatedly: according to the type of object, the watermark will be inserted into selected objects respectively. As to a given subset, we select a special bit position in a value and reset it to one bit of

watermark information according to the object's id. We can repeatedly insert watermark information to the attribute values and propagate to other attributes.

- ⑤ Verify whether the watermarked data is among the bounds, or satisfy the semantic constraints and structural constraints.

Algorithm 1. shows the watermark insertion process:

---

**Algorithm 1. AddWatermark()**


---

```

AddWatermark (κ , attri, MSA, mark_info, λ , ζ , repeatnum, bounds)
 marks [] = ECC(mark_info[]); //preprocess original watermark
 for each tuple $t \in R$ do
 for each sub-object $\in t$ do
 id=Hash($\kappa \circ MSA(A_i | A_i \circ A_j \circ \dots \circ A_v) \circ$ sequence of sub-object);
 //calculate the id of each sub-object
 subset \leftarrow sort all the sub-objects by id and divide into
 subsets;
 subset_boundaries[] \leftarrow the id of the 1st sub-object;
 for ($i=0$; $i < \text{length}(\text{marks}[])$; $i++$)
 mark(subset i);
 If (not Constraints.Satisfied(new_data, bounds)) then
 //watermarked data is not satisfied with the constraints
 {false_array[] \leftarrow id; Rollback}
 else commit;

 subroutine mark(subset i)
 for each sub-object \in subset i do
 if (id % $\lambda == 0$) //watermark the sub-object
 { bit_index j = id % ζ ; //watermark the jth bit of the
 value
 switch(geom1->type) //judge the type of the object
 case Point:
 pt = (Point)geom1; break;
 //get the address of the point
 case Line:
 {line = (Line)geom1; npoint = line->npoints;
 //get the number of points in the line
 i = id % npoint; //watermark the ith point
 pt = (point)line->point[i]; break;}
 case Polygon:
 {npoint = geom1->npoints;
 //get the number of points in the polygon
 i = id % npoint;}
 if (IsFirstOrEndRing(polygon, i))
 //the point is the start or end of the ring
 pt1 = GetAnotherPoint(polygon, i);
 //get the other terminal
 IsTwoPoint = true; break;
 }
 ModifyPoint(pt, j, k) //the jth bit of coordinates are set
 to k
 If (IsTwoPoint)
 ModifyPoint(pt1, j, k);}

```

---

### 3.3 Watermark Detection and Error Correction

Firstly, we calculate each sub-object's id and reconstitute subsets with the assistant of subset\_boundaries[] array. Then, to detect one bit of mark in each subset by the first majority-voting. Last, to use majority-voting for the second time in all subsets to get the detected watermark. The majority-voting method can eliminate some small errors. Thus the mechanism can be more robust to attacks.

As to detect\_mark', using BCH decoding method to correct the errors in the watermark and to generate final watermark. The subroutine **Compare()** is to verify the similarity between original watermark and the detected one. If the similarity is larger than a threshold, we can suspect piracy.

---

#### Algorithm 2. DecWatermark()

```

Detect(attribute, MSA, κ, λ, ζ, repeatnum, subset_boundaries[])
 for each tuple t ∈ R do
 for each sub-object ∈ t do
 id=Hash(κ ∘ MSA(Ai | Ai ∘ Aj ∘ ... ∘ Av) ∘ sequence of
 sub-object); //calculate the id of each sub-object
 for (i=0; i<length(marks[]); i++)
 {if subset_boundaries[i]< id <subset_boundaries[i+1]
 then subseti ← the sub-object; }
 //put each sub-object into its corresponding subset
 for (i=0; i<length(marks[]); i++)
 temp[] ← extract (subseti);
 //extract watermark in each subset
 detect_mark' = majority_voting(temp[]');
 //decide the correct mark bit by majority-voting
 detect_mark = Decode(detect_mark);
 //error correcting by BCH
 compare(mark_info, detect_mark) to verify piracy;

 subroutine extract(subseti) return number
 for (n=0; n<subset_size; n++)
 if ((id mod λ == 0) && (id not in false_array)) then
 bit_index j = id mod ζ;
 temp[]← the last jth bit of the marked point;
 return majority_voting(temp[])

```

---

## 4 Implementation and Evaluation

The system implementation is in Redhat 9.0 with PostgreSQL 7.2 supported by PostGIS. C language is used here as to preserve the original coding style of PostgreSQL

**Table 2.** Experimental environment

CPU	Pentium 2.4 GHz
RAM	SAMSUNG 512 MB
OS	Linux RedHat9.0
DBMS	PostgreSQL+PostGIS
SERVER	Apache2+PHP

and can call the inner functions of PostgreSQL in order to attain high performance. The experimental environment is shown in Table 2.

#### 4.1 Overhead of the System

We ran two experiments to assess the computational cost of watermark insertion and detection. Performance was measured in elapsed time. Each experiment was repeated 20 times and the overhead ratios were computed from the average of individual trials.

The first experiment evaluated the cost of inserting a watermark. We tried the worst case by setting  $\lambda$  to 1. In this case, the watermarking algorithm will read and mark all the tuples. However, on average, half of the tuples will already have the correct value for the mark. Therefore, we expect that watermarking will update only half tuples. We compare these latencies to the time required to read all the tuples and update half tuples. The comparison yielded a ratio of 1.92, showing a rather small overhead of 92% incurred by watermarking. This overhead is due to the cost of computing hash values needed to determine the mark for individual tuples and the cost of BCH coding.

The second experiment assessed the cost of detection. We again chose the worst case by setting  $\lambda$  to 1 and by choosing the sample size for detecting the watermark to be the entire relational table. The experiment compared the time required to detect marks in all the tuples against the time required to simply read all the tuples. The comparison yielded a ratio of 10.12. This cost seems a little high, however, we should point out that the major consuming of the cost in detection is the computation of one way hash functions needed to determine the presence of the mark for each tuple and BCH decoding procedures.

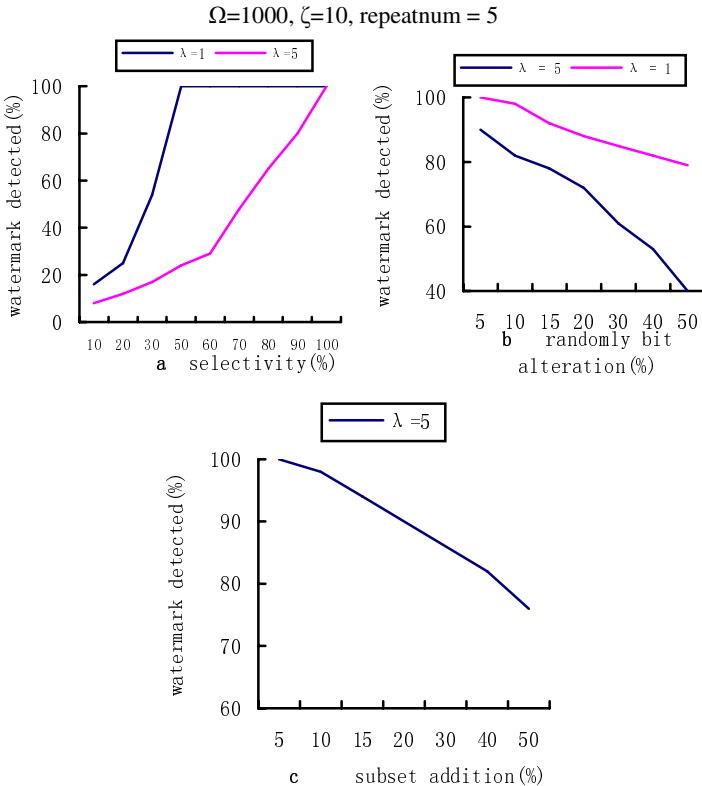
These results indicated that our algorithms have adequate performance to allow for their use in real world applications.

#### 4.2 Experiments Evaluation

The function of watermarking in Geo-WDBMS is tested based on large amount of geometry data which is faced with malicious attacks. We classify all kinds of attacks into three classes: selection attack, alteration attack, and addition attack. We carry experiments on a vector map with 1000 tuples: attack the watermarked data with the three methods and give the robustness evaluations of the watermark. Definition of the parameters used in the watermarking algorithms can be referred to section 3.2.

##### 4.2.1 Selection Attack

Selection attack is such a kind of attack that selects part of watermarked data aiming to delete partial watermarks. Fig. 3.a shows the results when watermarked vector map was attacked by subset selection. When  $\lambda=1$ , we can detect the watermark by the ratio of 100% in approximately 50% of data and for  $\lambda=5$  we can detect by the ratio of 25% in 50% of tuples. This demonstrates that higher the ratio of watermarked data is, the smaller the fraction of tuples is needed, and better the selectivity of subset is, the higher the ratio of detected watermark is.



**Fig. 3.** (a)Watermark detected in selection attack (b) Watermark detected in alteration attack (c) Watermark detected in addition attack

#### 4.2.2 Alteration Attack

The attacker may randomly change some data in order to erase the watermark. Fig. 3.b shows the results of our experiment, when  $\lambda=1$ , we randomly change 50% of tuples by resetting 1 bit in a value and can detect approximately 79% of watermark. when  $\lambda=5$ , we can also detect 40% watermark by changing 50% tuples randomly. The results indicate that with the increasing of  $\lambda$ , the amount of watermark information decreases and the robustness weakens.

#### 4.2.3 Addition Attack

The attacker may add some tuples to the watermarked table. But this form of attack almost has little impact to the watermark. In the experiment, the results are similar when  $\lambda=1$  and  $\lambda=5$ , which shows that our algorithm is very robust to such addition attack. And Fig. 3.c indicates that the watermark survives at least 75% when 50% of tuples are added.

#### 4.2.4 Other Attacks

Above analyses are the main attacks threatening watermarks. However, there are some other attacks, for example: mixture attack, additive attack and collusion attack. Mixture

attack is to combine the above three attacks and the robustness to it is analogous to above analyses. Additive attack is to add another watermark into the watermarked data, that is, there are two copies of watermark in the data. In the paper [6], Agrawal points that the probability of the collision of watermark bits is 0.1%, so thus bit-resetting watermark is robust to additive attack. Collusion attack is that two or more users carrying different versions of the same data to collude, compare their data, find a part of embedded watermarks, and make an unauthorized copy by removing or disabling the original marks. However, this is beyond our scope of focus and there are papers concerned on the issue particularly.

## 5 Conclusion and Future Work

In this paper, we have studied the technology of watermarking geographical databases for verifying piracy. And we proposed a new watermark framework with error correction mechanism, which pays much attention to correcting the errors in the detected watermarks by using BCH coding method. Further more, Geo-WDBMS has been built based on PostgreSQL, which has the function of watermarking numeric data and geometry data. The method of watermarking Geo-data is designed specially for geometry data. And the experiments showed that the watermarked digital map is robust to various attacks.

In the future, on the one hand, with the research of multi-representation maps in our lab, we will pay more attention to enhancing the robustness of the system by improved algorithms. On the other hand, in order to trace the illegal digital copies, we will combine watermark with TSA (Trusted Spotting Agent) technology and thus we can protect the copyrights of digital products on the network.

## References

1. Zhang Yong, Zhao Dong-ning, Li De-yi: Digital Watermarking Techniques and Progress. Journal of PLA University of Science and Technology, 2003, 4(3), pp.1–5
2. Zhang Li-he, Yang Yi-xian, Niu Xin-xin, Niu Shao-zhang: A Survey on Software Watermarking. Journal of Software, 2003, 14(2), pp. 268–277
3. Radu Sion, Mikhail Atallah, Sunil Prabhakar: Key Commitment in Multimedia Watermarking, Jan. 2002 (CERIAS TR 2002-30), <http://www.cs.stonybrook.edu/~sion/>
4. M. A. Bishr. Geospatial Digital Rights Management with focus on Digital Licensing of GML datasets. Thesis of the International Institute for Geo-information Science and Earth Observation. March, 2006.
5. Mark A. Masry: A Watermarking Algorithm for Map and Chart Images. the Proceedings of the SPIE Conference on Security, Steganography and Watermarking of Multimedia Contents VII, January 2005.
6. Rakesh Agrawal, Jerry Kiernan: Watermarking Relational Databases. Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002, pp.155–166
7. Radu Sion, Mikhail Atallah, Sunil Prabhakar: Rights Protection for Relational Data. Proceedings of ACM SIGMOD 2003, San Diego, pp. 98–109
8. Radu Sion, Mikhail Atallah, Sunil Prabhakar: On Watermarking Numeric Sets. Proceedings of the Workshop on Digital Watermarking IWDW 2002, Seoul, Korea

9. Hank Wallace: error detection and correction using the bch code.  
<http://www.aqdi.com/bch.pdf>
10. Xu Zhou, Duyan Bi: Use Digital Watermarking to Protect GIS Data by Chinese Remaindering. Journal of Image and Graphics, Vo l. 9, No. 5, 2004
11. Kyi Tae Park, Kab Il Kim, Hwan Il Kang, and Seung Soo Han: Digital Geographical Map Watermarking Using Polyline Interpolation. PCM 2002, LNCS 2532, pp. 58–65, 2002.
12. Ohbuchi Ryutarou, Ueda Hiro, Endoh Shu: Robust watermarking of vector digital maps. Proceedings of IEEE Conference on Multimedia and Expo 2002 (ICME 2002), Lausanne , Switzerland , 2002, 8.
13. P.Shankar: On BCH Codes over Arbitrary Integer Rings. IEEE Trans. Inform. Theory, Vol. IT-25, pp. 480–483, July 1979
14. Lijun Zhang, Zhigang Cao and Chunyan Gao: Application of RS-coded MPSK Modulation Scenarios to Compressed Image Communication in Mobile Fading Channel. Proceedings 2000 52nd IEEE Vehicular Technology Conference, VTS-Fall VTC. 2000, Volume: 3, 2000 pp. 1198–1203
15. A.Ambroze, G.Wade, C.Serdean, M.Tomlinson, J.Stander, and M.Borda: Turbo Code Protection of Video Watermark Channel. IEEE Proceedings-Vision, Image and Signal Processing, Volume: 148, Issue: 1, Feb 2001 pp. 54–58

# TinTO: A Tool for the View-Based Analysis of Streams of Stock Market Data

Andreas Behrend, Christian Dorau, and Rainer Manthey

University of Bonn, Institute of Computer Science III

Roemerstr. 164, D-53117 Bonn, Germany

{behrend,dorau,manthey}@cs.uni-bonn.de

**Abstract.** TinTO is an experimental system aiming at demonstrating the usefulness and feasibility of applying conventional SQL queries for analyzing a wide spectrum of data streams. As application area we have chosen the analysis of streams of stock market data, mainly because this kind of application exhibits sufficiently many of those characteristics for which relational query technology can be reasonably considered in a stream context. TinTO is a technical investor tool for computing so-called technical indicators, numerical values calculated from a certain kind of stock market data, characterizing the development of stock prices over a given time period. In contrast to other approaches, TinTO computes indicator values directly over the database by means of SQL queries/views.

## 1 Technical Analysis of Stock Market Data

Technical analysis is concerned with the prediction of future developments of stock market prices. In contrast to fundamental analysis, it is solely based on the trading history while ignoring the nature of the company or commodity in question. Technical analysis uses so-called technical indicators, numerical values derived from the past development of prices of a certain stock. In principle, indicators are functions applied to the price history of a certain stock and a point in time. A technical analyst is usually interested in the change of indicator values over a certain time period in order to predict the future price development of the stock to which this indicator has been applied. The *simple moving average of the typical price* (SMATP) is an example of a rather simple technical indicator, defined as follows:

$$\begin{aligned} \text{SMATP}_n(S,D) &:= (\text{TP}(S,D) + \text{TP}(S,D-1) + \dots + \text{TP}(S,D-n+1))/n \\ \text{TP}(S,D) &:= (\text{high}(S,D) + \text{low}(S,D) + \text{close}(S,D))/3 \end{aligned}$$

Here,  $\text{TP}(S,D)$  denotes the typical price of stock S at day D which is the mean of the highest, lowest, and closing price of stock S at day D.  $\text{SMATP}_n(S,D)$  then represents the unweighted mean of the typical stock price of S for the last n days. The parameter n is provided by the user and usually ranges between 10 and



**Fig. 1.** Main window of TinTO

200. Moving averages such as the SMATP are used to smooth out short-term fluctuations, thus highlighting longer-term trends or cycles in the underlying price history. When the stock price rises above the current SMATP value, this is interpreted as the beginning of a positive price trend and, thus, may serve as a buy signal.

## 2 The TinTO System

In its most basic form, TinTO is not much more than a nice interface for visualizing answers to analytical queries evaluated over a relational database of stock data (cf. Figure 1) together with a simple query editor. The data stored are timestamped prices of stocks contained in a portfolio freely configurable from a wide range of stocks traded worldwide. TinTO is a Visual Basic (VBA) application based on MS Access. As a frontend it uses the shareware visualizer ChartDirector [2], a tool supplying a VBA library of well-established methods for drawing financial charts. Even though ChartDirector comes along with a wide spectrum of built-in technical indicators (computed by VBA functions) we extended the tool by a means to specify arbitrary indicators as predefined SQL queries (i.e., as views), evaluated directly over the underlying database. The values of these query-based indicators are visualized by ChartDirector in the same way as those computed by the tool's own indicator functions. By extending the tool with a simple SQL view editor we can offer a system for specifying new and modifying existing indicator definitions in an extensible manner. The system thus extended will be called ChartDirector++ in the following. At present we experiment with some 30 view-based indicators making use of various SQL features. Even though some interesting indicators cannot be expressed due to restrictions in SQL in general or its Access dialect in particular, e.g. lack of recursion, SQL has already proved sufficiently expressive in most cases.

Even though a considerable degree of analysis is reachable this way, hardly any streaming is involved yet, unless one already considers e.g. the sequence of daily closing prices of stocks as a very low frequency "stream". However, there are so-called intraday trading strategies which need to access a high frequency stream. For example, an intraday strategy could employ a time duration D of 3 seconds for computing the value of  $\text{SMATP}_n(S, D)$ .

The crucial step towards proper stream management in TinTO consisted in the addition of a simple VBA script automatically downloading a record of characteristic values per stock in the port-folio at regular intervals and appending the downloaded data to those already present in the database. The source we use is <http://finance.yahoo.com> while the stocks to be included and the frequency of download can be freely configured by the TinTO user. The software component thus realized - which we will call StockGrepper in the following - generates a data stream pulled from a permanent data source on demand as long as the script is active. This pulling approach enables us to control duration and frequency of data generation on the stream according to our needs.

### 3 Continuous Online Analysis of Stock Data

As soon as ChartDirector++ and StockGrepper are combined, a simple form of online analysis of stocks over continuously changing data can be performed. There are two independent processes in operation: StockGrepper produces data and appends it to the database, ChartDirector++ consumes these data and evaluates the prefabricated technical indicator definitions as queries over the stream data accumulated in the database, thus covering newly arrived as well as historical data depending on a chosen time span. In an initial setting, the producer process works autonomously with a certain download frequency, whereas the consumer process is started manually and by need only.

However, as soon as ChartDirector++ is turned into a semi-automated tool, too, autonomously updating its presentation of the selected chart in regular intervals, affairs get more intricate. The SQL definitions of the technical indicators are now turned into continuous queries to be repeatedly evaluated over a continuously growing database of stock prices [1]. There is a certain analysis frequency which ought to be at most as fast as the download frequency, but will probably be considerably slower in most realistic trading scenarios. As long as the time needed for evaluating each indicator query is less than the difference between two subsequent "ticks" of the pull mechanism, synchronization can easily be obtained by simply alternating between download and analysis. If the download frequency were increased below the time needed for refreshing the chart over the database, a buffering strategy would be required, decoupling the producer process from the consumer process. At present, we do not yet work with a buffer but keep the download frequency slow enough for being able to deal with "naive" synchronization.

Another problem apart from synchronization is the control of the sheer amount of data pulled from the stream. Even with a moderate download frequency (like once per minute) and only a handful of stocks in the portfolio, the number of entries will already challenge a mini-DBMS like Access and even a commercial-strength DBMS to its limits, so that an archiving strategy would be needed. At present, a simple archiving script is provided as another component in TinTO, copying selected entries from the main stream table into special archive tables at configurable intervals (e.g. once per hour and/or once per day).

## 4 Efficient Delta View-Based Analysis of Data Streams

Our present experiments with the setting of TinTO outlined so far serves the purpose of identifying limits of efficiency of technical indicators expressed as continuous SQL queries against a dynamically growing repository of stored stream data. Depending on the choice of download and analysis frequency as well as the power of the DBMS used such limits will be easily reached sooner or later. However, for quite a wide range of realistic trading strategies even a limited system like TinTO is already sufficiently powerful to master the data size and stream frequency needed to perform the required analytical tasks in SQL.

In a further extension of TinTO we try to further push the efficiency limits of view-based technical analysis by computing answers to continuous queries incrementally rather than to re-evaluate each indicator definition each time a refreshment is triggered. For this purpose, we record stock data which have newly arrived since the last evaluation of the indicator in special delta tables, representing adjacent sliding windows over the monitored stream of stock data. The view-based definitions of the indicators are then transformed into so-called "delta views" which restrict computation to determining the effect of the newly arrived delta facts only. Using delta tables and delta views is not new. In fact, delta techniques have been proposed in many contributions to the deductive database literature for efficiently performing "traditional" DB tasks such as integrity checking over views and maintenance of materialized views [3]. In our group, we contributed to the development of delta techniques e.g. within the international IDEA project during the 1990s (cf. [4]).

In principle, delta views can be automatically compiled from the original views. At present, we do not yet have a full-fledged delta compiler for arbitrary SQL views at hand, but perform our experiments with hand-compiled delta views only. Increase in efficiency of continuous query evaluation obtained so far in many cases encourages us to continue along this line. However, translating transformation methods defined mostly in a Datalog context in the literature on SQL is a non-trivial task (not yet mastered by relational DBMS vendors even for integrity and view materialization purposes), so that we did not yet invest too much effort in delta compiler construction before not having convincing evidence for the usefulness of applying delta techniques in the streaming context.

## References

1. SHIVNATH BABU, JENNIFER WIDOM: *Continuous Queries over Data Streams*. SIGMOD Record 30(3): 109-120 (2001)
2. *Chart Director*. <http://www.advsofteng.com> (09.10.2006)
3. ASHISH GUPTA, INDERPAL SINGH MUMICK: *Materialized Views: Techniques, Implementations, and Applications*. The MIT Press (1999)
4. ULRIKE GRIEFAHN, THOMAS LEMKE, AND RAINER MANTHEY: *Chimera Prototyping Tool: User Manual*. Technical Report IDEA.DE.22.O.006, ESPRIT Project 6333 (IDEA), 1996

# Danaïdes: Continuous and Progressive Complex Queries on RSS Feeds

Wee Hyong Tok, Stéphane Bressan, and Mong-Li Lee

School of Computing  
National University of Singapore  
`{tokwh, steph, leeml}@comp.nus.edu.sg`

**Abstract.** RSS (Really Simple Syndication) is a format used for the publication and syndication of web content. While several frameworks, techniques and algorithms have been proposed and studied for the processing of complex queries on data streams, current RSS reader and aggregator software and services do not propose advanced query facilities.

We designed and implemented a prototype RSS aggregator service, called *Danaïdes*, for the processing of complex queries on continuously updated RSS feeds and of progressively producing results.

We demonstrate the prototype and its several user-interfaces with a geographical application using geoRSS feeds. This work is a practical application of our research on progressive query processing algorithms for data streams.

## 1 Introduction

RSS (Really Simple Syndication) is an XML format used for the publication and syndication of web content. Users subscribe to RSS feeds using RSS readers and aggregators. Although readers and aggregators need to pull and filter data from the RSS feeds at regular intervals, RSS technology implements web data streams.

Existing RSS reader and aggregator software and services provide at most basic keyword-based filtering and simple feed merging. These software and services do not yet support complex queries. Such a support however would enable the utilization of RSS feeds to their full potential of continuous data streams and motivate, in a virtuous circle, the production and consumption of data.

We have designed and implemented a prototype RSS aggregator service, called *Danaïdes*, capable of processing complex queries on continuously updated RSS feeds and of progressively producing results. Users subscribe their queries to the service in a dialect of SQL that can express structured queries, spatial query and similarity queries. The service continuously processes the subscribed queries on the referenced RSS feeds and, in turn, published the query results as RSS feeds. The user can read the result feed in a standard reader software or service or in a dedicated interface.

We demonstrate the prototype and its several user-interfaces with a geographical application using geoRSS feeds. This work is a practical application of our research on progressive query processing algorithms [1][2][3] for data streams.

## 2 Related Work

In [4], the authors describe how commercial databases can be used as a declarative RSS Hub offering structured query capabilities. Since RSS is an XML format it is also natural (yet beyond the scope of the proof of concept that this paper is contributing) to consider XQuery for the formulation of complex query on RSS feeds. In [5], the authors demonstrate the use of XQuery for the filtering and merging of RSS feeds from several blogs.

Whether supporting SQL or XQuery the query processing engines of the new aggregators that we propose must be capable of continuously processing data streams. The above mentioned proposals for complex query in RSS aggregation do not take into account the dynamic and continuous aspect of the RSS feeds. New algorithms are being developed for the processing of queries on data streams. The various algorithms proposed, from the XJoin [6] to the Rate-based Progressive Join (RPJ) [7], Locality-Aware Approximate Sliding Window Join [8], Progressive Merge Join [9] and our Result-Rate Based Progressive Join (RRPJ) [3], try and propose non-blocking solutions that maximize throughput. While [6,7,8] only consider relational data , our solution [3] and [9] can be easily applied to data in other data models.

As far as we know, this is the first proposal for a continuous query processing service for RSS feeds aggregation.

## 3 Scenario and Prototype

The availability of precise, instantaneous, seamless and effortless positioning with the Global Positioning System (GPS), Galileo and GSM triangulation coupled with or embedded in personal and professional portable devices, equipment and gadgets allows the geo-tagging of content created anytime anywhere. From the casual souvenir photographs of a tourist time-stamped, and geo-tagged with longitude, latitude and altitude, published on Flickr<sup>1</sup> to the critical earthquake monitoring data from the U.S. Geological Survey [10], geo-tagged data is commonly published as RSS feed (A specialization of RSS to publish geographical data is called GeoRSS [11]).

In this demonstration we show the processing of several complex queries on multiple GeoRSS feeds. We use data from the United States Geological Survey Earthquake Hazards Program [10]. We show, in particular, queries involving relational joins, spatial joins and similarity join (see Figure 1). Results are then delivered progressively to the user as a GeoRSS feed. The result feed can be viewed using any RSS reader or aggregator software or service. We use Internet Explorer<sup>2</sup>. The result feed can also be viewed on a 2D or 3D map. We use a visualization interface that we have developed, which uses Virtual Earth<sup>3</sup> [12]. Figure 2 illustrates these user interfaces.

---

<sup>1</sup> Flickr is a trademark of Yahoo! Inc.

<sup>2</sup> Internet Explorer is a trademark of Microsoft Corp.

<sup>3</sup> Virtual Earth is a trademark of Microsoft Corp.

Find pairs of earthquake alerts with the same title within 5.6 degree of both latitude and longitude.

```
SELECT *
FROM rss("http://earthquake.usgs.gov/eqcenter/recenteqlww/catalogs/
eqs1day-M2.5.xml") a, rss("http://earthquake.usgs.gov/eqcenter/
recenteqsww/catalogs/eqs7day-M5.xml") b
WHERE a.title = b.title and
 dist(a.geoLat, a.geoLong, b.geoLat, b.geoLong) < 5.6
```

**Fig. 1.** Sample Query

The screenshot shows the 'Danaïdes Results' page. It displays two items from an RSS feed. Item 1 is about a quake at Kachemak Bay State Park, Alaska, on December 15, 2006. Item 2 is about a quake in Glacier walking, December 15, 2006. A sidebar on the right shows filtering options for 'Displaying 30 / 30', 'Sort by: All', and 'Date Title'.

0) http://earthquake.usgs.gov/eqcenter/recenteqlww/Quakes/ak00073432.php, 150.8446,December 15, 2006 02:15:10 GMT,M 3.8, Southern Alaska,61.5684,http://localhost/myhols,H 3.12, Stay at lodge at Kachemak Bay State Park,December 15, 2006 05:10:11 GMT,-158.65,6.38917690561369

1) http://earthquake.usgs.gov/eqcenter/recenteqlww/Quakes/ak00073432.php, 150.8446,December 15, 2006 02:15:10 GMT,M 3.8, Southern Alaska,61.5684,http://localhost/myhols,H 3.13, Glacier walking,December 15, 2006 05:10:11 GMT,-

(a) RSS Result Output (Displayed in Internet Explorer 7)



(b) Virtual Earth Augmented with GeoRSS Result

**Fig. 2.** Various ways of visualizing results from *Danaïdes*

The *Danaïdes* prototype consists of a scanner and a query processing engine. The scanner periodically pulls data from RSS feeds. The query engine consists of physical algebra operators (e.g. hash join, similarity join, selection, and projection). It constructs a query plan, executes the plan and produces a RSS feed consisting of the results.

## 4 Conclusion

We propose to demonstrate an application of our prototype RSS aggregator, *Danaïdes*, to the querying of earthquake alerts. Our aggregator main characteristic is the support for the publishing of continuous and progressive complex queries on RSS feeds.

## References

1. Tok, W.H., Bressan, S.: Efficient and adaptive processing of multiple continuous queries. In: EDBT. (2002) 215–232
2. Tok, W.H., Bressan, S., Lee, M.L.: Progressive Spatial Join. In: SSDBM. (2006) 353–358
3. Tok, W.H., Bressan, S., Lee, M.L.: RRPJ : Result-Rate based Progressive Relational Join. In: DASFAA. (2007) (To be published)
4. Gawlick, D., Krishnaprasad, M., Liu, Z.H.: Using the Oracle database as a declarative RSS hub. In: SIGMOD. (2006) 722
5. Ivanov, I.: Processing RSS - <http://www.xml.com/pub/a/2003/04/09/xquery.html> (2003)
6. Urhan, T., Franklin, M.J.: XJoin: Getting fast answers from slow and bursty networks. Technical Report CS-TR-3994, Computer Science Department, University of Maryland (1999)
7. Tao, Y., Yiu, M.L., Papadias, D., Hadjieleftheriou, M., Mamoulis, N.: RPJ: Producing fast join results on streams through rate-based optimization. In: SIGMOD. (2005) 371–382
8. Li, F., Chang, C., Kollios, G., Bestavros, A.: Characterizing and exploiting reference locality in data stream applications. In: ICDE. (2006) 81
9. Dittrich, J.P., Seeger, B., Taylor, D.S., Widmayer, P.: Progressive merge join: A generic and non-blocking sort-based join algorithm. In: VLDB. (2002) 299–310
10. <http://earthquake.usgs.gov/>: (U.S. geological survey earthquake hazards program)
11. <http://www.georss.org>: (GeoRSS:: Geographically encoded objects for rss feeds)
12. <http://www.microsoft.com/virtualearth/>: Microsoft virtual earth (2006)

# OntoDB: It Is Time to Embed Your Domain Ontology in Your Database

Stéphane Jean, Hondjack Dehainsala, Dung Nguyen Xuan, Guy Pierra,  
Ladjel Bellatreche, and Yamine Aït-Ameur

LISI/ENSMA - Poitiers University - France  
`{jean,hondjack,nguyenx,pierra,bellatreche,yamine}@ensma.fr`

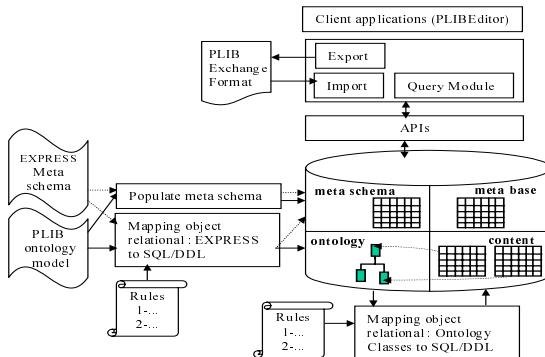
**Abstract.** This demonstration presents OntoDB, a prototype that allows to store explicitly in the database not only the data, but also the conceptual model defining the structure of data and the domain ontology representing the meaning of data. The demonstration illustrates three main functionalities of OntoDB: (1) a storage of a domain ontology and database content in the same repository, (2) the possibility of querying databases at ontology level, and (3) an automatic integration of heterogeneous data sources referencing/extending the same domain ontology.

## 1 Introduction

Traditionally, the process of database application design goes through a chain of three major steps: conceptual, logical and physical. The conceptual model (CM) is the core of the application development. Its basic constructs (entity, relationship between entities) are associated with semantics which can be understood intuitively by designers and users. This model is then translated into a logical model. Once this translation done, CM is usually discarded from the design chain. Consequently, application semantics described by this CM may be lost. Other work on ontology was undertaken in knowledge modeling [4]. Contrary to CM, an ontology aims to *describe* in a consensual way the whole knowledge of a domain. This description is agreed and shared by domain experts allowing them to understand each other. When such an ontology exists, the process of database design no longer needs to create completely new conceptualization, but it just *needs to extract or to specialize from the domain ontology pieces of information that are relevant for the application to be designed*. We call this approach *ontology-based modeling*. Recently, Sugumaran et al. work [7] shows how domain ontology can be used to assist in the generation of complete and consistent database conceptual design. Several approaches and systems were proposed to store in the same database, data and the ontologies describing their meanings [12]. In this demonstration, we present one of these systems named OntoDB [3]. By storing the conceptual model defining the structure of data, OntoDB is the only one to follow the ontology-based modeling approach. Moreover, we have shown in [3] that it outperforms other systems for a set of queries.

## 2 OntoDB Components

OntoDB represents explicitly: (1) ontologies, (2) data structures, (3) data, (4) the links between the data and their schema and (5) the link between schema and the ontology. Before defining the architecture of our prototype, we present the three objectives assigned to our architecture model: (1) it shall support an automatic integration and management of heterogeneous populations whose data, schemas and ontologies are loaded dynamically, (2) it shall support evolutions of the used ontologies (adding new classes, new properties, etc.) and of their population schemas, and (3) it shall offer data access, at the ontology level, whatever the type of the used DataBase Management System (DBMS) (relational, object-relational or object). Taking in account these objectives, our architecture is composed in four parts, where part 1 (meta base or system catalog) and part 2 (content) are traditional parts available in all DBMSs, and part 3 (ontology) and part 4 (meta schema) are specific to OntoDB (figure 1).



**Fig. 1.** System Architecture

*Ontology part* allows to represent ontologies in the database. When ontology model is object oriented and the target DBMS is relational, its logical schema is defined using an object/relational mapping. The *meta schema part* records the ontology model into a reflexive meta model. For the ontology part, the meta schema part plays the same role as is played by the *meta base* in traditional DBMSs. Indeed, this part allows: (1) a generic access to the ontology part, (2) support of evolution of the used ontology model, and (3) storage of different ontology models (OWL, PLIB [6], etc.). The link between ontology, meta base, and content parts is established using a global universal identifier mechanism associated to classes and properties of ontologies.

## 3 System Implementation

This section shows the implementation of each part of OntoDB. APIs to access OntoDB and modules of client applications are also described (see figure 1).

**1. OntoDB.** OntoDB is implemented on PostgreSQL7.4 and the (multilingual) PLIB ontology model (POM) is specified in EXPRESS (a formal OO modeling language associated with an environment similar to Meta Object Facility). To implement ontology part, POM has been mapped to a logical schema by a program generator. It is based on defined transformation rules between EXPRESS concepts and SQL/DDL. The logical schema of the meta schema part is also generated automatically by re-using an object relational generator. Concretely, the generator receives as input parameter an EXPRESS meta model of EXPRESS and returns a set of tables representing the meta model. Then the meta schema part is populated with the POM and with itself as data. To define the content part logical schema, the database designer selects a subset of the ontology that represents its CM and then the logical schema is generated by another object relational mapping which takes the CM as input parameter.

**2. Import module.** Like OWL in XML, the POM allows to represent and exchange both ontologies and ontologies instances in EXPRESS exchange format. The *import module* allows to read a population of the POM (ontology + data) and to store it in the database. If the ontology already exists in the database, it can be potentially updated (version management) and its instances will be automatically integrated in the existing population of ontology instances.

**3. Export module.** It is dual of the import module. It allows to extract a subset of an ontology from the classes of the ontology in the database, with or without the associated content. This module combined with import module allows to automatically *migrate* instances of a database to another.

**4. Ontology and content edition module.** It allows to dynamically display and create classes and their properties in the ontology and to dynamically create and visualize objects of these classes. Since each data is associated to an ontological element which defines its meaning, it becomes possible to access the data through their meaning. The interface offered by this module, called *PLIBEditor* is appropriate to any ontology and any population of classes.

**5. APIs.** Almost all the modules of figure 11 access OntoDB using a three layered API that we have implemented. The first layer is a generic API defined at the meta level (all parameters are strings). It allows to create instances independently of any model in any part of the database, but without any user control. The second layer specific to the POM, is composed of java classes generated automatically, one for each POM entity. This API calls functions of the first API. The last layer is also generated automatically to represent ontology classes as java instances. Note that most of the developed programs, modules and API are not POM-specific but they may be specialized for any ontology model after its description in the EXPRESS language.

The OntoDB architecture requires new query functionalities more those offered by traditional languages like SQL2003. We developed a query language, called OntoQL that allows to query data in terms of concepts of the ontology using its expressive power (multilingual, polymorphism, etc.) [5]. Moreover, it provides a way to extract a part of the ontology with its associated content.

## 4 The Scenario to Be Demonstrated

Our demonstration is based on an shared ontology (SO) defining concepts of the LMD (License, Master, Doctorate) university course. The LMD system has been established in order to harmonize diplomas in the *European Union*. The scenario of our demonstration follows 5 mains steps: **(1) Description of the used shared ontology:** the process of editing ontologies and the characteristics of the POM are shown as well. Our ontology editor (PLIBEditor) is used as client application. **(2) Definition of local ontologies from SO:** two different databases (representing two universities) specialize the SO to define the particular concepts existing in their own course. **(3) Extraction of conceptual models derived from the ontology:** the conceptual models of the two different databases are designed from their local ontologies. Several instances of students are described (using the ontology concepts) and inserted in both databases. **(4) Automatic integration of information:** shows the process of students data integration of the two universities. **(5) Query processing:** a set of retrieval queries using our QBE interface is executed on the integrated data. We show specific queries expressed in different natural languages on data involving ontology concepts that use the expressive power of the ontology model (like inheritance, composition, ...).

Additionally, in our demonstration, we present PLIBEditor that allows to manage (create, delete, import, export, query, etc.) ontologies and ontology-based data stored in OntoDB. The use of the OntoDB architecture to conceptually design a database using domain ontologies is also demonstrated. For more details, refer to our Web site <http://wwwplib.ensma.fr/plib/demos/ontodb/>, some "flash" demonstrations and snapshots are presented.

## References

1. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ics-forth rdfsuite: Managing voluminous rdf description bases. In *2nd International Workshop on the Semantic Web (SemWeb'01)*, 2001.
2. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *Proceedings of the First International Semantic Web Conference (ISWC'02)*, pages 54–68, July 2002.
3. H. Dehainsala, G. Pierra, and L. Bellatreche. Ontodb: An ontology-based database for data intensive applications. In *Proceedings of Database Systems for Advanced Applications, 12th International Conference (DASFAA'07) (to appear)*, 2007.
4. T. Gruber. A translation approach to portable ontology specification. *Knowledge Acquisition*, 7, 1993.
5. S. Jean, Y. Aït-Ameur, and G. Pierra. Querying ontology based database using ontoql (an ontology query language). In *Proceedings of OTM Confederated International Conferences (ODBASE'06)*, pages 704–721, 2006.
6. G. Pierra. Context-explication in conceptual ontologies : The PLIB approach. In *Proceedings of Concurrent Engineering (CE'03)*, pages 243–254, July 2003.
7. V. Sugumaran and V. C. Storey. The role of domain ontologies in database design: An ontology management and conceptual modeling environment. *ACM Transactions on Database Systems (TODS)*, 31(3):1064–1094, September 2006.

# Author Index

- Aßfalg, Johannes 586  
Achtert, Elke 152  
Aguilar-Saborit, Josep 6  
Ailamaki, Anastassia 374  
Aït-Ameur, Yamine 1119  
Alkobaisi, Shayma 624  
Anisetti, Marco 943  
Anutariya, Chutiporn 924  
Ardagna, Claudio A. 943
- Bae, Wan D. 624  
Bailey, Thomas 624  
Balke, Wolf-Tilo 551  
Behrend, Andreas 1110  
Bellandi, Valerio 943  
Bellatreche, Ladjel 497, 1119  
Bernardoni, Elisa 943  
Bertino, Elisa 188  
Bhowmick, Sourav S. 275, 793, 819  
Boey, S.H. 225  
Böhlen, Michael 1058  
Böhm, Christian 152  
Bressan, Stéphane 43, 994, 1115  
Bühmann, Andreas 349  
Burns, Randal 374  
Byun, Ji-Won 188
- Cai, Jing 884  
Cao, Jiaheng 1098  
Carminati, Barbara 410  
Caroprese, Luciano 459  
Chang, Edward Y. 522  
Cheema, Muhammad Aamir 863  
Chen, Arbee L.P. 300  
Chen, Bo 807  
Chen, Gang 576, 1050  
Chen, Jidong 611  
Chen, Lei 313, 509, 962  
Chen, Lijun 652  
Chen, Qiming 386  
Cheng, James 753, 768, 891  
Cheng, Jiefeng 18  
Chhieng, Van M. 598  
Choi, Byron 793
- Choi, Heeseok 955  
Chun, Jonghoon 398  
Chung, Chin-Wan 715, 975  
Costa, António C. 262  
Cui, Bin 563, 652, 664, 898
- Damiani, Ernesto 943  
Dang Ngoc, Tuyêt Trâm 1001  
Dehainsala, Hondjack 497, 1119  
Dewey Jr., C. Forbes 275, 819  
Ding, Bolin 18, 850  
Dong, Jinxiang 576  
Dong, Yisheng 1028  
Dorau, Christian 1110  
Du, Xiaoyong 386
- Fan, Yushun 1062  
Feng, Jianhua 834  
Ferrari, Elena 410  
Foo, Jun Jie 472
- Gao, Aiqiang 918  
Gao, Hong 1036  
Gao, Jun 652  
Gao, Yan 912  
Gardarin, Georges 1007  
Gebski, Matthew 176  
Greco, Sergio 459  
Gruenwald, Le 981  
Gu, Shiwen 912  
Güntzer, Ulrich 551  
Guo, Jinhua 905
- Han, In Kyu 930  
Hara, Takahiro 884  
Härder, Theo 349  
Hartmann, Sven 1070  
Hasan, K.M. Azharul 288  
Higuchi, Ken 288  
Hoksza, David 361  
Hsu, Wynne 31, 164  
Hu, Haibo 611  
Hu, Xiaohua 115  
Huang, Min 1098  
Huang, Shangteng 213

- Hvasshovd, Svein-Olaf 249  
 Hwang, Hyun Sik 1075  
 Hwang, Seung-won 539  
 Jain, Ankur 522  
 Jamard, Clement 1007  
 Jean, Stéphane 1119  
 Jeong, Jin-Woo 485  
 Jiang, Nan 981  
 Jing, Liping 115  
 Jørgensen, Peter Sune 1058  
 Kalashnikov, Dmitri V. 325  
 Kamra, Ashish 188  
 Kang, Hyun-Ho 447  
 Kao, Ben 103  
 Ke, Yiping 891  
 Kim, Deok-Hwan 1024  
 Kim, Jae-Myung 447  
 Kim, Jaehoon 1054  
 Kim, Kyu Il 1075  
 Kim, Sang-Wook 201  
 Kim, Seongjin 955  
 Kim, Seon Ho 624  
 Kim, Seung-Woo 201  
 Kim, Ung Mo 1075  
 Kitajima, Shinya 884  
 Kitsuregawa, Masaru 1, 703, 1020  
 Ko, Hyuk Jin 1075  
 Koh, Judice L.Y. 164  
 Kolltveit, Heine 249  
 Kriegel, Hans-Peter 152, 337, 586  
 Kröger, Peer 152, 586  
 Kunath, Peter 337, 586  
 Kwon, Yongjin 140  
 Lafón-Gracia, Néstor 6  
 Lai, Caifeng 611  
 Lam, Kai Tak 164  
 Lamarre, Philippe 237  
 Larriba-Pey, Josep-L. 6  
 Lau, Ho-Lam 1013  
 Le, Dung Xuan Thi 994  
 Lee, Chun-Hee 975  
 Lee, Dong-Ho 485  
 Lee, Dongwon 949  
 Lee, Ig-hoon 398  
 Lee, Jihyun 715, 975  
 Lee, Jongwuk 539  
 Lee, Ju-Hong 1024  
 Lee, Mong-Li 31, 43, 164, 1115  
 Lee, Rubao 1086  
 Lee, Sang-goo 398  
 Lee, Sang-Won 447  
 Lee, Sang Ho 930  
 Lee, Soowon 930  
 Leonardi, Erwin 793  
 Li, Chen 422  
 Li, Guoliang 688, 834  
 Li, Hanyu 31  
 Li, Jianhua 912  
 Li, Jianzhong 1036  
 Li, Juanzi 1066  
 Li, Ling 31  
 Li, Ninghui 188  
 Li, Xian 936  
 Li, Xiaoyan 1050  
 Li, Yongnian 676  
 Li, Zude 676  
 Lian, Xiang 962  
 Liao, Jia 127  
 Liao, Zhining 912  
 Lin, Dan 563  
 Lin, Jiexi 975  
 Lin, Xuemin 863  
 Ling, Tok Wang 807  
 Link, Sebastian 1070  
 Liu, Tianxiao 1001  
 Liu, Yunfeng 664, 898  
 Lo, Chia-Hao 870  
 Lofi, Christoph 551  
 Lohman, Guy M. 3  
 Loo, K.K. 103  
 Lu, An 891  
 Luan, Hua 386  
 Luo, Bo 949  
 Ma, Yiming 1045  
 Madeira, Henrique 262  
 Madria, Sanjay Kumar 703  
 Malik, Tanu 374  
 Manthey, Rainer 1110  
 Mehrotra, Sharad 325, 1045  
 Meng, Weiyi 936  
 Meng, Xiaofeng 434, 611, 936  
 Min, Jun-Ki 715, 969  
 Molinaro, Cristian 459  
 Mondal, Anirban 703  
 Moon, Mikyeong 955  
 Moon, Yang-Sae 79

- Morishita, Shinichi 781  
 Muntés-Mulero, Victor 6  
 Müller-Gorman, Ina 152  
 Na, Gap-Joo 447  
 Natwichai, Juggapong 1041  
 Nehme, Rimma V. 637  
 Ng, Michael 115  
 Ng, Wilfred 753, 768, 891, 1013  
 Ni, Yongzhi 386  
 Nishio, Shojiro 884  
 Nuray-Turan, Rabia 325  
 Nutanong, Sarana 876  
 Okabe, Yasuo 140  
 Ooi, Beng Chin 688  
 Orlowska, Maria E. 1041  
 Özsü, M. Tamer 807  
 Papadomanolakis, Stratos 374  
 Park, Kyung-Wook 485  
 Park, Myung-Jae 975  
 Park, Sang-Ho 1024  
 Park, Sanghyun 201  
 Park, Seog 1054  
 Parker, D. Stott 740  
 Peng, Wen-Chih 870  
 Peng, Zhaohui 1032  
 Peng, Zhiyong 1098  
 Pierra, Guy 497, 1119  
 Pokorný, Jaroslav 361  
 Prakash, Sandeep 819  
 Pryakhin, Alexey 586  
 Qian, Gang 1028  
 Qian, Weining 55  
 Qin, Lu 850  
 Quiané-Ruiz, Jorge-Arnulfo 237  
 Rahayu, Wenny 994  
 Rao, Fangyan 1062  
 Reale, Salvatore 943  
 Renz, Matthias 337, 586  
 Rundensteiner, Elke A. 637  
 Saito, Taro L. 781  
 Seah, Boon-Siew 793  
 Serres, Olivier 975  
 Shi, Baile 225  
 Shim, Junho 398  
 Shirani-Mehr, Houtan 422  
 Shou, Lidan 576, 1050  
 Sinha, Ranjan 472  
 Skopal, Tomáš 361  
 Song, Guojie 664, 898  
 Song, Jungsuk 140  
 Song, Shaoxu 313  
 Song, Yang 275  
 Su, Yu-Chi 300  
 Sun, Xingzhi 1041  
 Takakura, Hiroki 140  
 Tan, Kian-Lee 410, 1050  
 Tan, Wei 1062  
 Tang, Jie 1066  
 Tang, Shiwei 918  
 Taniar, David 994  
 Tanin, Egemen 876  
 Terada, Tsutomu 884  
 Theodoratos, Dimitri 727  
 Tok, Wee Hyong 43, 1115  
 Travers, Nicolas 1001  
 Trinh, Thu 1070  
 Tsuji, Tatsuo 288  
 Ungrangsi, Rachanee 924  
 Valduriez, Patrick 237  
 Vieira, Marco 262  
 Wang, Bin 962  
 Wang, Guoren 127, 509  
 Wang, Ping 576  
 Wang, Shan 67, 386, 1032  
 Wang, Wei 225, 988  
 Wang, Xiaodan 374  
 Wang, Xiaoling 988  
 Wang, Yitong 1020  
 Wang, Yuan-Fang 522  
 Wang, Zhenhua 509  
 Wang, Zhihui 225  
 Widjanarko, Klarinda G. 793, 819  
 Won, Jung-Im 201  
 Wong, Raymond K. 176, 598  
 Wu, Ling 898  
 Wu, Xiaoying 727  
 Wu, Yi-Hung 300  
 Wuwongse, Vilas 924  
 Xia, Zhonghang 905  
 Xiao, Yan 949  
 Xiao, Zhen 434

- Xie, Kunqing 664, 898  
Xin, Junchang 509  
Xing, Guangming 905  
Xu, Jianliang 434, 611  
Xuan, Dung Nguyen 1119  
  
Yang, Donghua 1036  
Yang, Dongqing 563, 652, 664, 898, 918  
Yang, Huei-You 870  
Yang, Hung-chih 740  
Yang, Weijia 213  
Yang, Xiaochun 422  
Yang, Zhenglu 1020  
Ye, Xiaojun 676  
Yeh, Laurent 1007  
Yeom, Keunhyuk 955  
You, Gae-won 539  
Yu, Bei 688  
Yu, Byunggu 624  
Yu, Ge 127  
Yu, Jeffrey Xu 18, 91, 850  
Yuan, Yidong 863  
  
Zhan, Jiang 67  
Zhang, Bo 127, 988  
Zhang, Jing 1066  
Zhang, Jun 1032  
Zhang, Rong 55  
Zhang, Rui 876  
Zhang, Xiaodan 115  
Zhang, Xiaoyi 509  
Zhang, Yong 834  
Zhao, Jiakui 652  
Zhao, Peixiang 91  
Zheng, Baihua 664  
Zhou, Aoying 55, 988  
Zhou, Lizhu 688, 834  
Zhou, Minghong 1086  
Zhou, Minqi 55  
Zhou, Xiang 1098  
Zhou, Xiaofang 127  
Zhou, Xiaohua 115  
Zhu, Jun 1062  
Zhu, Zhenzhou 807  
Zimek, Arthur 152