Computing Structural Statistics by Keywords in Databases

Lu Qin, Jeffrey Xu Yu, Lijun Chang

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

***Abstract*—Keyword search in *RDB*s has been extensively studied in recent years. The existing studies focused on finding all or top-k interconnected tuple-structures that contain keywords. In reality, the number of such interconnected tuple-structures for a keyword query can be large. It becomes very difficult for users to obtain any valuable information more than individual interconnected tuple-structures. Also, it becomes challenging to provide a similar mechanism like group-&-aggregate for those interconnected tuple-structures. In this paper, we study computing structural statistics keyword queries by extending the group-&-aggregate framework. We consider an *RDB* as a large directed graph where nodes represent tuples, and edges represent the links among tuples. Instead of using tuples as a member in a group to be grouped, we consider rooted subgraphs. Such a rooted subgraph represents an interconnected tuple-structure among tuples and some of the tuples contain keywords. The dimensions of the rooted subgraphs are determined by dimensional-keywords in a data driven fashion. Two rooted subgraphs are grouped into the same group if they are isomorphic based on the dimensions or in other words the dimensional-keywords. The scores of the rooted subgraphs are computed by a user-given score function if the rooted subgraphs contain some of general keywords. Here, the general keywords are used to compute scores rather than determining dimensions. The aggregates are computed using an SQL aggregate function for every group based on the scores computed. We give our motivation using a real dataset. We propose new approaches to compute structural statistics keyword queries, perform extensive performance studies using two large real datasets and a large synthetic dataset, and confirm the effectiveness and efficiency of our approach.**

I. Introduction

Keyword search on relational databases (*RDB*s) has been extensively studied. It allows users to query *RDB*s using keywords. Most of the existing studies focused on finding interconnected structures among tuples via foreign key references in a relational database that contain the keywords [1], [12], [10], [19], [20], [22], [3], [14], [15], [6], [8], [9], [5], [16], [23].

Take the *DBLP* dataset (http://www.informatik.uni-trier.de/~ley/db/) as an example to be stored in an *RDB* with 4 relations: Author(AID, Name, Affiliation), Paper(PID, Type, Title, CID), Conference(CID, Name, Year, Location), and Write(AID, PID). In the Author relation, an author is identified by AID, and is with a name and an affiliation. In the Paper relation, a paper is identified by PID, and has a type which can be either journal or conference. If a paper is a conference paper, its CID value refers to a conference in the Conference relation. A Conference is with a name, year, and location. The Write relation specifies the write relationships between authors and papers. In the most recent *DBLP* dataset we use in our study, there are 750,000 authors, among them we obtained affiliation information for 40,000 authors. Consider a keyword query {"keyword", "search", "graph"}. The existing approaches may find one among many interconnected structures such that an author Jim writes a paper that contains "keyword search" in its title and his co-author writes another paper that contains "graph" in its title. In reality, the number of such interconnected structures to be returned can be large. Thus, it becomes very difficult for users to identify any additional valuable information. In this paper, instead of finding interconnected structures among tuples, we study how to compute statistics on the interconnected tuple-structures using keywords.

Consider five keyword queries against the *DBLP* stored in *RDB*. (Q_1) Which conference is good for SQL query optimization? (Q_2) Which author is an expert on keyword search on graphs? (Q_3) Which author in which year has most papers about graph pattern mining? (Q_4) In which year, which conference is best for information retrieval on the web? (Q_5) In year 2007, which university has most papers about random walk on graphs? All these queries do not ask for what the individual interconnected structures look like. Instead, all these queries ask for some statistics. Take Q_1 as an example. Q_1 is to compute some statistics regarding conferences about SQL query optimization. In other words, the main question of Q_1 is on conferences, and the statistics to be collected regarding conferences are based on SQL query optimization. To the best of our knowledge, there does not exist any keyword search approach which can find valuable statistical information to answer the five queries.

In this paper, we study computing structural statistics for keyword queries by extending the group-&-aggregate computing framework. Recall that in an RDBMS tuples are grouped into the same group if they have the same attribute values in the user-specified dimensional attributes (or simply dimensions), and an SQL aggregate function is used to aggregate the numerical attribute values of the tuples in the same group. The four main factors in group-&-aggregate computing are tuples, dimensions, a numerical attribute to which an SQL aggregate function will be applied. In our study, we consider an *RDB* as a large directed graph where nodes represent tuples, and edges represent the links among tuples. There is an edge from a tuple t_i to another tuple t_j if a foreign key defined on t_i references to the primary key defined on t_j.

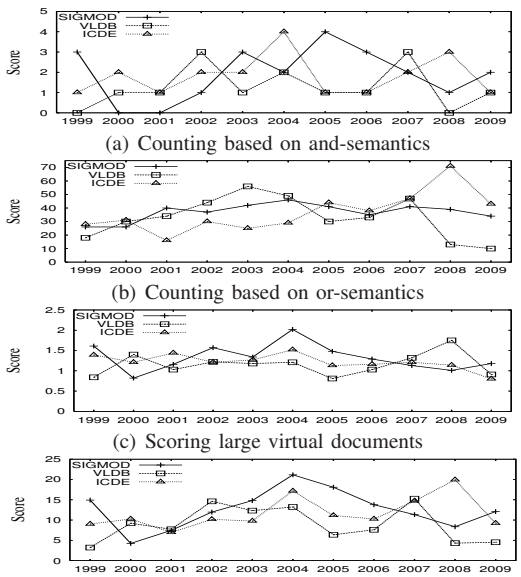
978-1-4244-8960-2/11/$26.00 © 2011 IEEE

We extend the group-&-aggregate computing framework as follows. (1) Instead of using tuples as a member in a group to be grouped, we consider rooted subgraphs as virtual tuples. Such a rooted subgraph represents an interconnected structure among tuples and some of the tuples contain keywords. (2) Regarding dimensions, it is worth noting that one of the main advantages of keyword queries is to find hidden interconnections among tuples in an *RDB* no matter where the keywords may appear or how the keywords appear. In other words, the dimensions in the conventional group-&-aggregate computing become data-driven in keyword search contexts. We introduce new dimensions which are determined by dimensional-keywords (a subset of user-given keywords). Let t_γ be a virtual tuple rooted at tuple t_γ. We use a set of dimensional-keywords to determine a dimensional object called Dtree rooted at t_γ where its leaf nodes contain all the dimensional-keywords. Two virtual tuples rooted at t_γ and t'_γ are grouped into the same group if their dimensional objects are isomorphic. (3) Regarding the numerical attribute to which an aggregate function is applied, we extract a Gtree from a virtual tuple t_γ using a set of general-keywords, and we apply a score function (α) to score the extracted Gtree as a virtual document, and then we aggregate such scores in a group using an SQL aggregate function (β).

The five queries can be specified using keywords as follows: {"conference", "SQL", "query", "optimization"} for Q_1, {"author", "keyword", "search", "graph"} for Q_2, {"author", "year", "pattern", "mining", "graph"} for Q_3, {"year", "conference", "information", "retrieval", "web"} for Q_4, and {"2007", "university", "random", "walk", "graph"} for Q_5. Here, the underlined keywords are dimensional-keywords, and the remaining keywords are general-keywords. Let the score function (α) be the tree level ranking function used in *SPARK* [19], and the aggregate function be $\beta = sum$. We list three answers with highest scores for the five queries returned by our system below.

- Q_1: (SIGMOD, 340.1), (VLDB, 274.5), (ICDE, 268.0).
- Q_2: ("Benny Kimelfeld", 31.9), ("Yehoshua Sagiv", 23.7), ("Yannis Papakonstantinou", 18.8).
- Q_3: ("Xifeng Yan", 2005, 8.2), ("Wei Wang", 2005, 6.8), ("Jiawei Han", 2007, 6.7).
- Q_4: (2004, SIGIR, 66.9), (2008, CIKM, 57.1), (2007, SIGIR, 46.1).
- Q_5: (2007, "Carnegie Mellon University", 12.0), (2007, "University of California", 8.7), (2007, "University of Waterloo", 7.8).

We further explain the meaning of α and β functions using another query: in which conference and which year, there are most papers about data integration (Q_6). This query can be formulated as a structural statistics keyword query {"conference", "year", "data", "integration"}. All the dimensional objects (Dtrees) contain both a conference name and a year, using the two dimensional-keywords. Such a dimensional object may start from a paper tuple, t_p, which links to a conference tuple t_c. The conference tuple t_c contains "conference" in its Name attribute, and contains a year value because there is an attribute named Year in the Conference relation, which the



(a) Counting based on and-semantics

(b) Counting based on or-semantics

(c) Scoring large virtual documents

(d) Scoring individually followed by aggregation

Fig. 1. {"conference", "year", "data", "integration"}

"year" keyword matches. In this simple application, the two keywords "data" and "integration" may appear in the title of the paper, and may appear as a part of the conference name. The related virtual document (Gtree) for the same paper tuple t_p is an interconnected tuple-structure from t_p that may contain either "data", or "integration", or both. We plot the answers in Fig. 1 using different α score functions, and different β aggregate functions. All the four figures in Fig. 1 shows scores for the three major database conferences, SIGMOD, VLDB, and ICDE from 1999-2009.

Fig. 1(a) and Fig. 1(b) show the scores using simple counts. Fig. 1(a) simply shows how many papers in a conference in a year that has both "data" and "integration" in the titles. It misses the papers that contain only one keyword. For example, in year 2001, there are three SIGMOD papers that contain only "integration" but not "data". Fig. 1(b) simply shows how many papers in a conference in a year that has either "data", or "integration", or both in the titles. It may over-count. For example, in year 2003, there are only three VLDB papers containing "integration", but 53 VLDB papers containing "data". Fig. 1(c) shows the TF-IDF score (α) by treating all the papers that contain either "data", or "integration", or both in their titles as a large virtual document for a conference in a year. Such approach is better than simple counts. But it may give some information which causes confusion. Consider a conference in two different years, y_1 and y_2. In year y_1, there are two papers, each of the two papers contain "data" and "integration" once. In year y_2, there is one paper that contains "data" twice, and one paper that contains "integration" twice. They have the same counts for the two keywords in y_1 and y_2. Obviously, the conference should have a higher score in y_1 than y_2. For example, in year 2008, there are three VLDB papers that contain "integration", but none of them contain both "data" and "integration". In year 2007, there are three VLDB papers that contain both "data" and "integration". But in Fig. 1(c), VLDB 2007 is scored lower than VLDB 2008. In

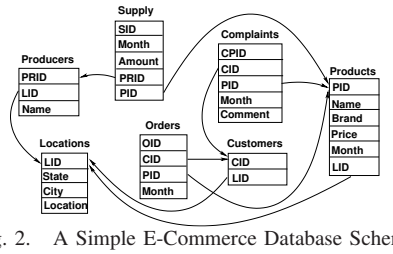


Fig. 2. A Simple E-Commerce Database Schema

CPID	CID	PID	Month	Comment
cp_1	c_1	p_1	February 2008	The monitor's quality is bad
cp_2	c_1	p_2	March 2008	It is not good for the computer
cp_3	c_2	p_2	March 2008	It is too small
cp_4	c_3	p_3	April 2008	I'm not satisfied for the color
cp_5	c_4	p_1	June 2008	The quality should be improved
cp_6	c_5	p_3	June 2008	I don't like the monitor
cp_7	c_3	p_4	May 2008	My monitor is broken
cp_8	c_2	p_5	May 2008	It is a little expensive

(a) Complaints

LID	State	City	Location
l_1	Illinois	Chicago	No.34, Road 3
l_2	Illinois	Chicago	No.53, Street 10
l_3	Nevada	Las Vegas	KM Tower, Road 9
l_4	California	Los Angeles	Eg Building, Street 33
l_5	California	Orange	No.121, Street 4
l_6	Washington	Tiger	No.113, Street 5

(b) Locations

CID	LID
c_1	l_1
c_2	l_2
c_3	l_3
c_4	l_4
c_5	l_4

(c) Customers

PID	Name	Brand	Price	Month	LID
p_1	Computer V7011	Orange	$1000	Jan. 2008	l_5
p_2	Monitor 3500 Black	Tiger	$300	Feb. 2008	l_6
p_3	Computer E1003	Orange	$1300	Jan. 2008	l_6
p_4	Monitor K133 Large	Tiger	$100	Mar. 2008	l_5
p_5	Monitor K140	Tiger	$190	Mar. 2008	l_6

(d) Products

Fig. 3. Four Relations in the E-Commerce Database

our approach, we consider each paper as an individual virtual document, and use a TF-IDF based α function to score, and then sum up all the scores for a conference in a year. The results are shown in Fig. 1(d). With our approach, VLDB 2007 is scored higher than VLDB 2008.

The main contributions of this work are summarized below. First, we study a new structural statistics keyword query in an *RDB*. We extend the existing work on group-&-aggregate over attribute values in several ways based on keywords in a data-driven fashion. Second, we give a two-step approach to process a structural statistics keyword query using label-trees. A label-tree is obtained based on the schema information. We show how to avoid tree isomorphism testing, and how to share cost in processing a structural statistics keyword query, using the label-trees. Third, we performed extensive performance studies using two large real datasets and a large synthetic dataset, and confirmed the effectiveness and efficiency of our approach.

The remainder of the paper is organized as follows. In Section II, we discuss structural statistics keyword queries. In Section III, we outline our two-step approach to compute structural statistics. We discuss the two steps in Section IV and Section V, respectively. We discuss the related work in Section VI. We show our experimental studies in Section VII and conclude our paper in Section VIII.

II. Structural Statistics

Let $GS = \{R_1, R_2, \cdots\}$ be a relational database schema. Here, R_i in GS is a relation schema with a set of attributes. We call an attribute a text-attribute in the paper if the attribute

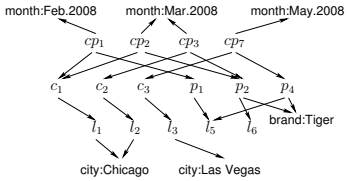


Fig. 4. A Database Subgraph for the E-Commerce Database

is defined on either string (char/varchar) or full-text domain. Keyword search is allowed on any text-attributes. A relation schema may have a primary key and there are foreign key references defined in GS. We use $R_i \rightarrow R_j$ to denote that there is a foreign key defined on R_i referring to the primary key defined on R_j. A relation on relation schema R_i is an instance of the relation schema (a set of tuples) conforming to the relation schema, denoted $r(R_i)$. A relational database (*RDB*) is a collection of relations.

Example 2.1: An E-Commerce database schema, GS, is shown in Fig. 2, which is modified based on TPC-H (http://www.tpc.org/tpch). It consists of seven relation schemas: Products, Customers, Orders, Locations, Producers, Complaints, and Supply. There are foreign key references denoted by $\rightarrow$. For example, there are two foreign key references from Complaints to two other relations Customers and Products, respectively. The text attributes include Name, Brand, Month, State, City, Location, and Comment. A part of the E-Commerce database is shown in Fig. 3 including four relations, namely, Complaints, Locations, Customers, and Products. □

We model an *RDB* over GS as a directed database graph $G_D(V, E)$. For easy discussion, we use two types of labeled nodes in the following discussions, $V = V_t \cup V_a$ for $V_t \cap V_a = \emptyset$. Here, V_t is the set of *tuple-nodes* for all tuples in *RDB*. A tuple-node, t_i, represents a tuple in $r(R')$ and is labeled with its relation name R'. V_a is the set of *attribute-nodes* for every distinctive pair of text-attribute and attribute value in *RDB*. An attribute-node is labeled $A_j : a_k$ where A_j is a text-attribute name and a_k is an attribute value. Accordingly, $E = E_{tt} \cup E_{ta}$ consists of two types of edges. E_{tt} is the set of edges among tuple-nodes in G_D, and an edge $t_i \rightarrow t_j$ in E_{tt} indicates that there exists a foreign key reference from t_i to t_j in *RDB*. E_{ta} is the set of edges from tuple-nodes to attribute-nodes. There exists an edge from a tuple-node t_i to an attribute-node $A_j : a_k$ in E_{ta}, if the tuple t_i has the attribute value a_k in the attribute A_j in *RDB*. We will use $V(G)$ and $E(G)$ to denote the set of nodes and the set of edges of G, respectively.

A part of the database graph G_D for the *RDB* in Fig. 3 is shown in Fig. 4. Here, cp_i, c_i, p_i, and l_i are tuple-nodes whose labels are Complaints, Customers, Products, and Locations, respectively. We omit the tuple-node labels in Fig. 4. There are also attribute nodes. For example, p_2 links to an attribute-node labeled "brand:Tiger". We say an attribute-node v contains a keyword k_i if k_i is contained in either the attribute name or the attribute value in the node label. In Fig. 4, an attribute-node labeled "brand:Tiger" contains the keywords "brand" and "tiger". We also say a tuple-node u contains a keyword k_i if there is a path from tuple node u to an attribute node v that contains k_i.

Virtual Tuple: We define a virtual tuple which is a tree representation of the maximum DAG (Directed Acyclic Graph) subgraph at a tuple-node t_γ in G_D. We denote such a virtual tuple as Vtuple, or explicitly Vtuple(t_γ) if it is rooted at a tuple-node t_γ. In Vtuple(t_γ), if two nodes u and v link to a node w in G_D, it will make an additional copy of w and let u and v link to different copies of the node w. All leaf nodes in Vtuple(t_γ) must be attribute-nodes. A Vtuple(t_γ) includes all information a tuple-node t_γ can reach, in a way to include all the nodes that t_γ can reach. As an example, u and v will be included in Vtuple(t_γ) if $t_\gamma \rightarrow u$ and $t_\gamma \rightarrow v$.

Dtree, Gtree, and DGtree: Given a set of keywords $\{k_1, k_2, \cdots\}$. A Dtree(t_γ) is a subtree of Vtuple(t_γ). Dtree(t_γ) must contain all dimensional-keywords by connecting to the attribute-nodes that contain the given dimensional-keyword(s). Dtree(t_γ) is minimal by which it means that the Dtree rooted at t_γ does not contain all the dimensional-keywords, if any node is deleted from the Dtree. A Gtree(t_γ) is also a subtree of Vtuple(t_γ) by removing all the subtrees rooted at a tuple-node that do not have any attribute-node containing general-keywords. A Gtree(t_γ) does not necessarily contain all the general-keywords. Given a set of keywords, there exists one Gtree(t_γ), but many distinctive Dtree(t_γ), by definition. A DGtree(t_γ) = Dtree(t_γ) $\cup$ Gtree(t_γ).

In this paper, we study a new structural statistics keyword query $Q = (Q_d, Q_g, \alpha, \beta)$ against an *RDB* over GS. It consists of two sets of keywords, namely Q_d and Q_g, for $Q_d \cap Q_g = \emptyset$, a score function α, and an aggregate function β. We call a keyword in Q_d and Q_g as a dimensional-keyword and a general-keyword, respectively. The two sets of keywords, Q_d and Q_g, together specify a set of trees $\mathcal{T}$ to be computed. A tree $T_i \in \mathcal{T}$ is a tree rooted at $t_\gamma = \text{root}(T_i)$, called a DGtree. It literally consists of two subtrees, a Dtree rooted at t_γ for Q_d, and a Gtree rooted at the same t_γ for Q_g, denoted as Dtree(T_i) and Gtree(T_i), respectively. The set of trees, $\mathcal{T}$, are grouped into different groups. Let T_i and T_j be two trees in $\mathcal{T}$. T_i and T_j belong to the same group if Dtree(T_i) is isomorphic to Dtree(T_j). Here, the tree isomorphism is defined over the labeled Dtrees. As an example, $cp_2 \rightarrow c_1 \rightarrow l_1$ and $cp_3 \rightarrow c_2 \rightarrow l_2$ in Fig. 4 are isomorphic to each other because they have the same structure and the same labels, Complaints $\rightarrow$ Customers $\rightarrow$ Locations. T_i and T_j may have different Gtrees, even though they are in the same group.

We allow a score function α to be any possible algebraic function based on TF-IDF, namely, $tf_w(T)$ and idf_w. We explain it below. Let $\mathcal{T} = \{T_1, T_2, \cdots\}$ be a set of DGtrees in the same group. We consider every Gtree(T_i) for $T_i \in \mathcal{T}$ as a virtual document, by merging all attribute names and attribute values in the tree into a multi-set. Then, $tf_w(T_i)$ is the number of times the keyword $w \in Q_g$ appears in the corresponding virtual document, and idf_w is calculated as follows.

$$idf_w = \frac{|\mathcal{T}|}{df_w(\mathcal{T}) + 1} \quad (1)$$

where $|\mathcal{T}|$ is the number of Gtrees in $\mathcal{T}$ in the group, and $df_w(\mathcal{T})$ is the number of Gtrees that contain the keyword $w \in Q_g$ in the group. The tree level ranking function used in *SPARK* [19] is such an algebraic function based on $tf_w(T)$ and idf_w. The α function is to be applied to Gtree(T_i) for $T_i \in \mathcal{T}$ to give such Gtree(T_i) a score. The aggregate function β aggregates the scores computed for DGtrees in the same group. An aggregate function can be any SQL aggregate functions (min, max, sum, avg, and $count$). The output for the group $\mathcal{T}$ is (T_A, ω), where T_A represents the Dtree for the group, and $\omega = \beta(\{\alpha(\text{Gtree}(T_1)), \alpha(\text{Gtree}(T_2)), \cdots\})$.

Consider $Q_d = \{$"city", "month"$\}$ and $Q_g = \{$"computer", "monitor"$\}$. Fig. 5 shows three pairs of Dtrees and Gtrees rooted at cp_1, cp_2, and cp_3, respectively, based on the *RDB* in Fig. 3. We omit the tuple-node labels in Fig. 5. A DGtree is a union of the Dtree and Gtree rooted at the same node. For example, the DGtree rooted at cp_1 is the union of the Dtree rooted at cp_1 and the Gtree rooted at cp_1 in Fig. 5(a). The Dtrees rooted at cp_2 and cp_3 are isomorphic to each other, because their labeled trees are isomorphic to each other. Note that $cp_2 \rightarrow c_1 \rightarrow l_1$ and $cp_3 \rightarrow c_2 \rightarrow l_2$ have the same structure with the same labels, Complaints $\rightarrow$ Customers $\rightarrow$ Locations. The corresponding DGtrees rooted at cp_2 and cp_3 belong to the same group. The Dtree rooted at cp_1 is not isomorphic to the Dtree rooted at cp_2.

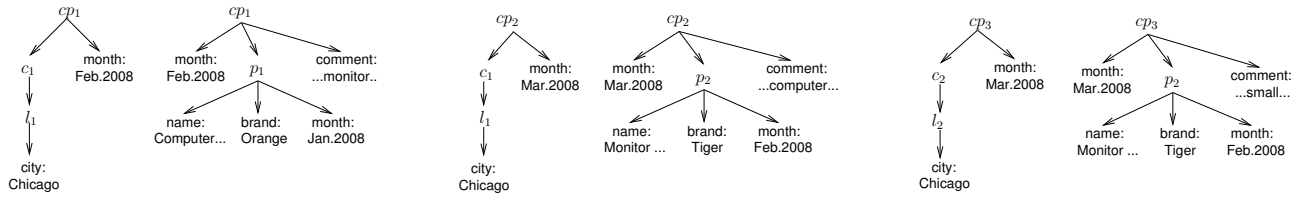
Algorithm 1 Structural-Statistics(GS, Q, G_D)

1: $\mathcal{L}$ = LD-Gen(GS, Q);
2: **for every** LD$_i \in \mathcal{L}$ **do**
3: LD-Eval(Q, LD$_i$, G_D);

III. Solution Overview

Given a structural statistics keyword query $Q = (Q_d, Q_g, \alpha, \beta)$ over an *RDB*, a naive solution is to first compute all Dtrees using Q_d, which can be done using any existing algorithm in the literature. Let $\mathcal{T}_A = \{T_{a_1}, T_{a_2}, \cdots\}$ be the set of non-empty Dtrees computed. Second, for each $T_{a_i} \in \mathcal{T}_A$, it expands from root($T_{a_i}$) to identify its Gtree using Q_g. Let $\mathcal{T}_O = \{T_{o_1}, T_{o_2}, \cdots\}$ be the set of corresponding Gtrees computed. Third, it computes $\alpha(T_{o_i})$ for each non-empty $T_{o_i} \in \mathcal{T}_O$. Fourth, it groups all Dtrees, T_{a_i} into groups, if the corresponding T_{o_i} is non-empty. Finally, it computes β for each group, and outputs the results. Such a naive solution is impractical for the following two main reasons. (1) The number of possible Dtrees and Gtrees can be very large. It is infeasible to compute. It is worth noting that all the existing solutions focus on finding top-k answers if they make use of ranking and allow some (not all) keywords to be contained in the answers. (2) It is costly to group Dtrees into groups even though tree isomorphism checking is polynomial.

In this paper, to compute a structural statistics keyword query, Q, we propose a two-step approach (Algorithm 1). In the first step, we generate a set of label-trees (LDs), denoted as $\mathcal{L} = \{\text{LD}_1, \text{LD}_2, \cdots\}$, such that every DGtree to be computed will conform to a unique LD. In the second step, we compute the structural statistics keyword query Q using $\mathcal{L}$. In brief, for every LD$_i$, we compute all DGtrees, denoted as $\mathcal{T}_i = \{T_{i_1}, T_{i_2}, \cdots\}$ that conform to LD$_i$, group all the trees in $\mathcal{T}_i$ into groups based on Dtree(T_{i_j}) and compute $\alpha(\text{Gtree}(T_{i_j}))$ for every $T_{i_j} \in \mathcal{T}_i$, and then compute β for



(a) Dtree (left) and Gtree (right) rooted at cp_1 (b) Dtree (left) and Gtree (right) rooted at cp_2 (c) Dtree (left) and Gtree (right) rooted at cp_3

Fig. 5. Dtrees and Gtrees

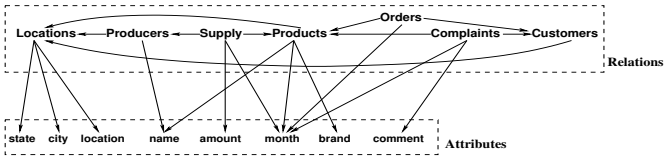
every group. The main idea behind label-trees is to avoid tree isomorphism checking and enumerating all possible DGtrees. In this section, we discuss the label-trees. The algorithms to generate all label-trees and to compute structural statistics keyword query using the label-trees will be discussed in the following sections.

First, we define a label-graph, $G_S(V, E)$, for keyword search for a database schema GS. Here, V is a set of nodes such that $V = V_R \cup V_A$, where V_R is a set of nodes labeled with relation names, called relation nodes, and V_A is a set of nodes labeled with attribute names for those text-attributes only. E is a set of edges such that $E = E_{RR} \cup E_{RA}$. An edge $R_i \rightarrow R_j$ appears in E_{RR} if there is a foreign key reference from R_i to R_j. An edge $R_i \rightarrow A_j$ appears in E_{RA} if R_i has an attribute A_j. The label-graph G_S for the E-Commerce database schema (Fig. 2) is shown in Fig. 6(a).

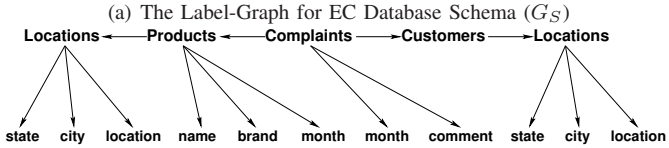
Second, we define a rooted tree for every relation R_i in G_S, denoted as $\mathsf{LV}(R_i)$, which is a labeled tree for all virtual tuples rooted at $t_\gamma \in r(R_i)$. $\mathsf{LV}(R_i)$ is a connected tree representation of the maximum DAG subgraph in G_S. In other words, if two nodes u and v link to a node w, it will make an additional copy of w and let u and v link to different copies of the node w. The reason of making a copy is explained below. Consider Fig. 2, the Location relation may be linked by two different relations, namely, Customers and Products, in the same LV(Complains). But the locations for customers are not necessarily to be the same locations for products. All leaf nodes in an LV must be attribute names. The LV(Complaints) for G_S (Fig. 6(a)) is shown in Fig. 6(b). An $\mathsf{LV}(R_i)$ is independent from any structural statistics keyword query.

Now consider a structural statistics keyword query $Q = (Q_d, Q_g, \alpha, \beta)$ against the data graph G_D using the label-graph G_S. We further give a specific LV for computing Dtrees. A label-tree $\mathsf{LD}_i(R_j)$ is a subtree of $\mathsf{LV}(R_j)$ rooted at R_j that contains at most $|Q_d|$ attribute-nodes as leaf nodes. The attribute-nodes in $\mathsf{LD}_i(R_j)$ possibly contain all the dimensional-keywords in Q_d. Given an LD tree, we use $\mathsf{att}(\mathsf{LD})$ to denote the set of attributes (or attribute-nodes) in the tree. Consider $Q = (Q_d, Q_g, \alpha, \beta)$, where $Q_d = \{$"city", "month"$\}$ and $Q_g = \{$"computer", "monitor"$\}$. There are two LDs, LD_1 and LD_2 in Fig. 6(c) and (d), which are subtrees of LV(Complaints) in Fig. 6(b). In this example, both keywords "city" and "month" are contained in the corresponding attributes in LD_1 and LD_2. There may exist other LDs if they have attributes that possibly contain both keywords "city" and "month".

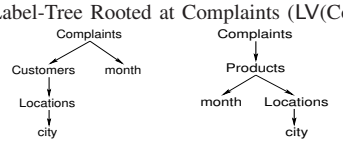
We observe that any $\mathsf{Dtree}(t_\gamma)$ conforms to one $\mathsf{LD}_k(R_i)$ if



(a) The Label-Graph for EC Database Schema (G_S)



(b) The Label-Tree Rooted at Complaints (LV(Complaints))



(c) LD_1 (d) LD_2

Fig. 6. Label-Graphs

t_γ represents a tuple in the relation $r(R_i)$. By conformation, we mean that $\mathsf{Dtree}(t_\gamma)$ is isomorphic to $\mathsf{LD}_k(R_i)$, if we remove all attribute values from the $\mathsf{Dtree}(t_\gamma)$. Note that a Dtree is a Vtuple where a tuple node is labeled with a relation name, and an attribute-node is labeled with a pair of attribute name and attribute value. We consider a $\mathsf{Dtree}(t_\gamma)$ as an instance of $\mathsf{LD}_k(R_i)$. We also observe that two different trees $\mathsf{Dtree}(t_i)$ and $\mathsf{Dtree}(t_j)$ are isomorphic to each other, if they have the identical $\mathsf{LD}_k(R)$ and have the same corresponding attribute values in $\mathsf{att}(\mathsf{LD}_k(R))$. The former says that two are structurally identical, and the latter says that the attribute values are the same. As an example, $\mathsf{Dtree}(cp_2)$ (left in Fig. 5(b)) and $\mathsf{Dtree}(cp_3)$ (left in Fig. 5(c)) belong to the same group (isomorphic), because the two Dtrees conform to the same LD_1(Complaints) in Fig. 6, and have the same attribute values "city:Chicago" and "month:Mar.2008". $\mathsf{Dtree}(cp_1)$ (left in Fig. 5(a)) is in a different group, because the attribute values do not match other attribute values. In addition, if two Dtrees, T and T' conform to two different LDs, they do not belong to the same group, because they have different structures.

Based on our observations, instead of checking tree isomorphism, we generate all possible LDs, and enumerate all Dtrees that conform to the same LD, and further group those Dtrees that conform to the same LD using attribute-value match, because we can ensure that they have the same tree structure already if they conform to the same LD.

IV. Generate All LDs

Our solution is as follows. First, we pre-compute all LVs for G_S because the set of LVs is query independent, which is straightforward. (Refer to Algorithm 2 which is

Algorithm 2 LV-Gen(G_S)

Input: The label-graph $G_S(V_R \cup V_A, E_{RR} \cup E_{RA})$.
Output: The set of all LVs.

1: $\mathcal{S} \leftarrow \emptyset$;
2: **for all** $R \in V_R$ **do**
3: add R with links to all its text-attributes into $\mathsf{LV}(R)$;
4: $\mathcal{Q} \leftarrow \emptyset$;
5: $\mathcal{Q}.enqueue(R)$;
6: **while** $\mathcal{Q} \neq \emptyset$ **do**
7: $R_1 \leftarrow \mathcal{Q}.dequeue()$;
8: **for all** $R_1 \rightarrow R_2 \in E(G_S)$ **do**
9: let R_2' be a copy of R_2;
10: add an edge in $\mathsf{LV}(R)$ from R_1 to R_2';
11: add links from R_2' to all its text-attributes;
12: $\mathcal{Q}.enqueue(R_2')$;
13: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathsf{LV}(R)\}$;
14: **return** $\mathcal{S}$;

Algorithm 3 LD-Gen(G_S, Q, $\mathcal{S}$)

Input: The label-graph G_S, a query $Q = (Q_d, Q_g, \alpha, \beta)$ and the set of LVs, $\mathcal{S}$.
Output: The set of all LDs.

1: $\mathcal{Q} \leftarrow \emptyset$;
2: **for each** keyword d_i in Q_d **do**
3: **for each** $e \in \mathsf{list}(d_i)$ **do**
4: **for each** $\mathsf{LV} \in \mathcal{S}$ **do**
5: **if** $e.Rel \in \mathsf{LV}$ **then**
6: $\mathsf{LV}.list_i \leftarrow \mathsf{LV}.list_i \cup \{e\}$;
7: **for each** $\mathsf{LV} \in \mathcal{S}$ **do**
8: **if** $\mathsf{LV}.list_i \neq \emptyset$ for any $1 \leq i \leq |Q_d|$ **then**
9: **for all** $(e_1, ..., e_{|Q_d|}) \in \mathsf{LV}.list_1 \times ... \times \mathsf{LV}.list_{|Q_d|}$ **do**
10: construct a minimal LD that only contain attribute-nodes $(e_1, ..., e_{|Q_d|})$;
11: $\mathcal{Q} \leftarrow \mathcal{Q} \bigcup \{\mathsf{LD}\}$;
12: **return** $\mathcal{Q}$;

self-explained.) Second, in order to efficiently generate all LDs for all possible dimensional-keywords, we construct an inverted index, called the dimensional inverted index (DII), using the names and values of the attributes in the *RDB*. The inverted index helps to find the attributes in a relation that a dimensional-keyword d_i matches. In detail, for each possible dimensional-keyword, w, in DII, there is a list of entries to describe the attributes in a relation the keyword w matches. We denote the list for w as $\mathsf{list}(w)$. Each entry $e \in \mathsf{list}(w)$ has three fields: e = (Type, Rel, Attr). Here, Type can be either Name or Value. When Type=Name, it means w is an attribute name. When Type=Value, it means w is an attribute value. Rel and Attr indicate the relation and the attribute that w matches. Consider the relational database shown in Fig. 3. For the dimensional-keyword "month", one entry in list("month") is (Name, Complaints, month). It suggests that "month" is an attribute name in the relation Complaints.

We design a new algorithm to generate all LDs using $\mathsf{list}(d_i)$ for $d_i \in Q_d$, as shown in Algorithm 3. The algorithm generates all LDs for a structural statistics keyword query. First, it collects information if a keyword d_i matches the relation nodes in each LV (line 2-6). Given the set of LVs $\mathcal{S}$, for each LV in $\mathcal{S}$, we use $\mathsf{LV}.list_i$ to maintain the set of candidate entries in $\mathsf{list}(d_i)$ that may contribute to generate LDs from LV. Second, in a for loop (line 7-10), it generates all LDs if they are matched by all keywords.

Theorem 4.1: *The time complexity of Algorithm 3 is* $O(\sum_{i=1}^{m} |\mathsf{list}(d_i)| \cdot |V_R(G_S)| + |\mathcal{Q}|)$*, where* $m = |Q_d|$ *and* $|\mathcal{Q}|$ *is the number of valid* LDs *generated.* □

Algorithm 4 LD-Eval-Naive(Q, LD, G_D)

Input: A query $Q = \{Q_d, Q_g, \alpha, \beta\}$, LD and a database graph $G_D(V, E)$.
Output: aggregates for all groups

1: $\mathcal{T} \leftarrow$ all Dtrees computed that conform to the LD;
2: $\mathcal{T}' \leftarrow \emptyset$; $\Gamma \leftarrow \emptyset$;
3: **for all** Dtree $T_i \in \mathcal{T}$ **do**
4: compute DGtree T_i' by expanding T_i in G_D;
5: $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{T_i'\}$;
6: **for all** $T_i' \in \mathcal{T}'$ **do**
7: $T_i'.score \leftarrow \alpha(T_i')$ by considering T_i' as a virtual document in $\mathcal{T}'$ w.r.t. Q_g.
8: let a_i be the attribute value in the attribute $A_i \in \mathsf{att}(\mathsf{LD})$ that contain d_i, for all $d_i \in Q_d$;
9: let γ be a group represented by $(a_1, a_2, \cdots, a_{|Q_d|})$;
10: $\gamma.score = \beta(\gamma.score, T_i'.score)$;
11: $\Gamma \leftarrow \Gamma \cup \{\gamma\}$ if $\gamma \notin \Gamma$;
12: **return** Γ;

Proof Sketch: The two nested loop (line 2-3) in Algorithm 3 needs to run $\sum_{i=1}^{m} |\mathsf{list}(d_i)|$ times for $m = |Q_d|$, and in every iteration, it runs in $O(|\mathcal{S}|) = O(|V_R(G_S)|)$ time (line 4-6). The total time complexity of line 2-6 is $O(\sum_{i=1}^{m} |\mathsf{list}(d_i)| \cdot |V_R(G_S)|)$. It runs $O(|\mathcal{S}|) = O(|V_R(G_S)|)$ times for line 7-10. Every time, line 8 needs $O(m)$ time for checking. All LDs generated are valid (line 9-10). The total time complexity of line 7-10 is $O(|V_R(G_S)| \cdot m + |\mathcal{Q}|)$, where $|\mathcal{Q}|$ is the number of all LDs. As a result, the total time complexity for Algorithm 3 is $O(\sum_{i=1}^{m} |\mathsf{list}(d_i)| \cdot |V_R(G_S)| + |V_R(G_S)| \cdot m + |\mathcal{Q}|) = O(\sum_{i=1}^{m} |\mathsf{list}(d_i)| \cdot |V_R(G_S)| + |\mathcal{Q}|)$. □

V. Evaluate ALL LDs

In this section, for a structural statistics keyword query $Q = (Q_d, Q_g, \alpha, \beta)$, we first give a naive approach followed by a new two-phase approach to compute structural statistics for all the groups under a given LD. As shown in Algorithm 1, we will compute all groups for every LD.

A. The Naive Approach

A naive approach to compute all groups under LD is to (1) compute all the Dtrees (for Q_d) and (2) determine the Gtrees (for Q_g). For (1), it can be done using an existing algorithm. For (2), we can expand from a Dtree computed to determine the corresponding DGtree. The naive approach is shown in Algorithm 4. In line 1, we generate all Dtrees that conform to LD, which can be done using an existing algorithm. For every Dtree generated, we compute the DGtree by expanding from each node in the Dtree to include all nodes it can reach and contain any of the keywords in Q_g. Line 6-11 is for group-&-aggregate. Since all the DGtrees has been computed, for each tree T_i', we compute $\alpha(T_i')$ (line 7) and identify its dimensional values that contain the corresponding keywords in Q_d (line 8). Then, we compute its group score using the β function (line 9-10). For simplicity, we use $\gamma.score = \beta(\gamma.score, T_i'.score)$, if β is min, max, sum and $count$. If β is avg we can do the same using both sum and $count$ values. The time complexity of Algorithm 4 is given in Theorem 5.1.

Theorem 5.1: *Let* $|P|$ *be the number of tuples in* P*,* N *be the number of nodes in* $\mathsf{LV}(R)$ *that generates* LD*,* A *be the average number of attribute nodes for each node in* $\mathsf{LV}(R)$*,*

$M = max\{|P| \text{ for } P \in V_R(G_S)\}$, *and* $m = |Q_d|$ *and* $n = |Q_g|$. *The time complexity for the naive algorithm to evaluate* LD *is* $O(M \cdot N \cdot (N + A^2 \cdot n + m))$. □

Proof Sketch: The time complexity consists of three parts: the time to compute all Dtrees in $\mathcal{T}$ (line 1), the time to compute the expanded DGtrees in $\mathcal{T}'$ (line 3-5) and the time for group-&-aggregate(line 6-11). For the first part, it needs $O(N^2 \cdot M)$ to compute all Dtrees. For the second part to expand each Dtree, it needs $O(N \cdot A)$ time to expand each tree. For each node expanded, it needs $O(n \cdot A)$ time to check whether it contains some of the $n = |Q_g|$ general-keywords. We totally have $O(M)$ trees to be expanded. So the time complexity for the second part to expand all Dtrees is $O(M \cdot N \cdot A^2 \cdot n)$. For the third part, there are at most M DGtrees in $\mathcal{T}'$ and for each DGtree, we need to compute its $m = |Q_d|$ dimensions following the path from the root node to the m attribute nodes. The time complexity for the third part is $O(M \cdot N \cdot m)$. Consequently, the time complexity for the naive algorithm is $O(M \cdot N^2 + M \cdot N \cdot A^2 \cdot n + M \cdot N \cdot m) = O(M \cdot N \cdot (N + A^2 \cdot n + m))$. □

There are mainly two drawbacks for the naive approach. It needs to compute all DGtrees before group-&-aggregate, which is costly. The same subtree will be computed several times in computing DGtrees and conducting group-&-aggregate.

B. A New Two-Phase Approach

We propose a new two-phase approach, namely, bottom-up followed by top-down, after pruning unnecessary nodes/edges from G_D that need to evaluate an $\mathsf{LD}(R)$. For a given $\mathsf{LD}(R)$ to be evaluated, let $\mathsf{LV}(R)$ be the labeled LV that generates $\mathsf{LD}(R)$. In the bottom-up phase, we collect statistics for TF-IDF, namely, $tf_w(T)$ and idf_w (refer to the discussions related to Eq. (1)) using general-keywords. The statistics collection is done in a bottom-up fashion, and will finish when it finally evaluates the relation R which is the root of $\mathsf{LD}(R)$. Let R' be the set of tuples in R that contain statistics for at least one general-keyword in Q_g. At the end of this phase, we have a set of $\{\mathsf{Gtree}(t_\gamma)\}$ for t_γ in R', and we compute all $\alpha(\mathsf{Gtree}(t_\gamma))$. In the top-down phase, we start from those tuples t_γ in R' and check if they certainly have a $\mathsf{Dtree}(t_\gamma)$ that contains all the required dimensional-keywords in Q_d in the attributes, as specified by the leaf attribute nodes in $\mathsf{LD}(R)$ that t_γ can reach. At the end of this top-down process, we compute β for all such t_γ in R' that have a valid $\mathsf{Dtree}(t_\gamma)$.

Below, we first discuss the pruning process before the two-phase. Given an LD, we consider whether a keyword $d_i \in Q_d$ has enough pruning power using $Pow(d_i, \mathsf{LD})$ which is computed as follows. Suppose $X(A, B)$ is the join selectivity in relation A that can join a tuple in relation B. Suppose the attribute in relation R_i in LD contains d_i, and assume P_i is the path from the root of LD to R_i. We have

$$Pow(d_i, \mathsf{LD}) = \frac{1}{R_i.c(d_i) \cdot \prod_{(A,B) \in P_i} X(A, B)} \quad (2)$$

Here $R_i.c(d_i)$ is the number of tuples in R_i that contain the keyword d_i in the specified attribute of R_i in LD. For any general-keyword $g_j \in Q_g$, suppose for any text-attribute A_i, $R(A_i)$ is the relation of A_i, and $P(A_i)$ is the path from the root of LD to the relation $R(A_i)$. We compute the pruning power of a general-keyword g_i, $Pow(g_i, \mathsf{LD})$ as follows.

$$Pow(g_i, \mathsf{LD}) = \frac{1}{\sum_{A_i \in \mathsf{LD}} R(A_i).c(g_i) \cdot \prod_{(A,B) \in P(A_i)} X(A, B)} \quad (3)$$

Based on Eq. (2) and Eq. (3), we decide whether a dimensional-keyword $d_i \in Q_d$ has enough power to prune, or in other words, it is cost-effective to reduce G_D. We sort all dimensional-keywords $d_i \in Q_d$ in decreasing order based on Eq. (2). If the pruning power of d_i is larger than the largest pruning power of all general-keywords in Q_g (Eq. (3)), then we use d_i to reduce G_D by removing all the tuple-nodes that cannot reach any attribute nodes containing d_i.

The algorithm is shown in Algorithm 5. First, we prune unnecessary nodes from G_D using Q_d if they have enough pruning power (line 5-7). Second, in the bottom-up phase, we compute trees in a sense to collect all information needed to compute α for every Gtree using Q_g. It is done from the leaf towards the root which is a tuple in $r(R)$ for the $\mathsf{LV}(R)$ that generates the given LD (line 9-25). Finally, in a top-down phase, we aggregate for each group based on Q_d (line 27-31).

We use the following notations. P denotes a relation node

Algorithm 5 LD-Eval(Q, LD, G_D)

Input: A query $Q = \{Q_d, Q_g, \alpha, \beta\}$, LD and a database graph $G_D(V, E)$.
Output: aggregates for all groups

1: $\Gamma \leftarrow \emptyset$;
2: let $\mathsf{LV}(R)$ be the LV that generates LD;
3: **for all** relation node $P \in \mathsf{LV}(R)$ **do**
4: $P.set \leftarrow \emptyset$;
5: **for all** $d_i \in Q_d$ sorted by decreasing order of $Pow(d_i, \mathsf{LD})$ **do**
6: **if** $Pow(d_i, \mathsf{LD}) > max_{g_i \in Q_g}(Pow(g_i, \mathsf{LD}))$ **then**
7: $G_D \leftarrow$ prune(G_D, d_i, LD);
8: *// The bottom-up phase*
9: **for all** relation node $P \in \mathsf{LV}(R)$ in the order from leaves to the root **do**
10: **for** $i = 1$ **to** $|Q_g|$ **do**
11: **for all** tuple-node $t_P \in P.contain(g_i)$ **do**
12: $t_P.cnt_i \leftarrow t_P.cnt_i + t_P.count(g_i)$;
13: $P.has_i \leftarrow P.has_i \cup \{t_P\}$;
14: $P.set \leftarrow P.set \cup \{t_P\}$;
15: **for all** child of P, C, in $\mathsf{LV}(R)$ **do**
16: **for all** node $t_C \in C.set$ **do**
17: **for all** node $t_P \in P$ such that $t_P \rightarrow t_C \in E(G_D)$ **do**
18: $P.set \leftarrow P.set \cup \{t_P\}$;
19: **for** $i = 1$ **to** $|Q_g|$ **do**
20: $t_P.cnt_i \leftarrow t_P.cnt_i + t_C.cnt_i$;
21: **if** $t_C.cnt_i > 0$ **then** $P.has_i \leftarrow P.has_i \cup \{t_P\}$;
22: **for all** tuple-node $t \in R.set$ **do**
23: **for** $i = 1$ **to** $|Q_g|$ **do**
24: $tf_{g_i}(t) \leftarrow t.cnt_i$; $df_{g_i}(\mathsf{LV}(R)) \leftarrow |R.has_i|$;
25: $t.score \leftarrow \alpha(t)$ using all $tf_{g_i}(t)$ and $df_{g_i}(\mathsf{LV}(R))$;
26: *// The top-down phase*
27: **for all** $t \in R.set$ **do**
28: let a_i be the attribute value in the attribute $A_i \in \mathsf{att}(\mathsf{LD})$ that contains d_i, for all $d_i \in Q_d$;
29: let γ be a group represented by $(a_1, a_2, \cdots, a_{|Q_d|})$;
30: $\gamma.score = \beta(\gamma.score, t.score)$;
31: $\Gamma \leftarrow \Gamma \cup \{\gamma\}$ if $\gamma \notin \Gamma$;
32: **return** Γ;

33: **Procedure** prune(G_D, d_i, LD)
34: let R' be the relation node in LD that links to an attribute for d_i;
35: let S be all R'-labeled tuple-nodes that link to an attribute-node containing d_i in G_D;
36: let S' be all nodes in G_D that can reach at least one node in S;
37: let S'' be all nodes in G_D that at least one node in S' can reach;
38: **return** the reduced graph of S'' in G_D;

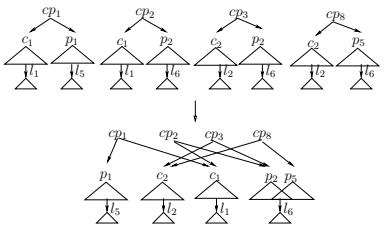


Fig. 7. Evaluate LD without generating all trees

in $\mathsf{LV}(R)$. $P.set$ denotes the set of P-labeled nodes in the Gtrees containing at least one keyword in Q_g, $P.has_i$ is a subset of $P.set$ including the nodes in the Gtrees that contain the keyword $g_i \in Q_g$. A tuple $t_P \in P.set$ is associated with a count, $t_P.cnt_i$, to record the number of times the keyword g_i appears in the Gtree rooted at t_P.

In Algorithm 5, we sort all dimensional-keywords $d_i \in Q_d$ in decreasing order based on Eq. (2) (line 5). If the pruning power is larger than the largest pruning power of all general-keywords in Q_g (Eq. (3)), then we can use it to reduce G_D (line 6-7). The prune procedure is shown in line 33-38, which returns a reduced subgraph of G_D.

The bottom-up phase is line 9-25. We visit all relation nodes $P \in \mathsf{LV}(R)$ in a bottom-up fashion. For each P-labeled node, t_P, we maintain the counting information for general-keywords. Here, t_P contains g_i if g_i is contained in one of its text-attributes. For t_P in $P.contain(g_i)$, $t_P.count(g_i)$ keeps the number of appearances of g_i in any text-attributes of t_P. Different from $t_P.count(g_i)$, $t_P.cnt_i$ keeps the number of appearances of g_i in the Gtree rooted at t_P. We add $t_P.count(g_i)$ into $t_P.cnt_i$ (line 12), and insert t_P into $P.set$ and $P.has_i$ (line 13-14). Suppose C is a child of P, and all the counting information for the Gtrees rooted at nodes in C has been computed, because we process relation nodes in $\mathsf{LV}(R)$ in the order from leaf nodes to the root node. We update counting information for nodes in P using the information computed for nodes in C (15-21). At the end of the bottom-up phase, the scores for all nodes in $R.set$ (the root of $\mathsf{LV}(R)$) can be computed using α (line 22-25).

The top-down phase is line 27-31. For each node t in $R.set$ computed in the bottom-up phase, we compute $\mathsf{Dtree}(t)$ that must have all the dimensional-keywords Q_d which must appear at specific attributes. We do it by searching along the path from t to the corresponding attribute nodes (line 28). Let γ be the group where t belongs to. We compute the aggregate function β using $t.score$ (line 30).

Cost Saving in the Bottom-up Phase: We discuss cost sharing in the bottom-up phase using an example. In Fig. 7, the upper part shows how the naive algorithm (Algorithm 4) does. There are 4 Gtrees, rooted at cp_1, cp_2, cp_3, and cp_8, being computed individually without cost sharing. In our new two-phase algorithm, we compute the $tf_w(T)$ and idf_w regarding $w \in Q_g$ from the leaf nodes to the roots. Suppose we have computed such $tf_w(T)$ and idf_w for the Locations-labeled nodes. The Locations-labeled node, l_6, is shared by both Gtrees rooted at p_2 and p_5, we pass the information computed on l_6 to p_2 and p_5 to achieve cost-sharing. In a similar way, the information on p_2 is further passed to cp_2 and cp_3, and thus the computational cost for computing the subtree rooted at p_2 is minimized.

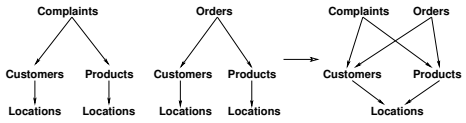


Fig. 8. Sharing Computational Cost cross LDs

Theorem 5.2: *Let $|P|$ be the number of tuples in P, N be the number of nodes in $\mathsf{LV}(R)$, $M = max\{|P| \text{ for } P \in V_R(G_S)\}$, and $m = |Q_d|$ and $n = |Q_g|$. The time complexity for the two-phase algorithm is $O((n+m) \cdot N \cdot M)$.* □

Proof Sketch: The first cost is the pruning step in line 4-7, it takes $O(M \cdot N)$ because each node in G_D is marked at most once. The second cost is the nested loops in line 9-21 in Algorithm 5. Obviously, the time complexity for line 10-14 is $O(M \cdot n)$, because $|P.contains(g_i)| \leq M$. For line 15-21, as line 19-21 need time complexity of $O(n)$, we show that it will visit at most $O(N \cdot M)$ nodes in total. Note that, for every child node of P, C, in the worst case, all nodes of P will be visited at most once in line 16-19. Suppose P has $C(P)$ child nodes. The total number of nodes visited for all nodes in the relation node P of $\mathsf{LV}(R)$ is at most $O(\sum_P C(P) \cdot |P|)$. Recall that $\mathsf{LV}(R)$ is a tree. We have $\sum_P C(P) < N$, consequently, $O(\sum_P C(P) \cdot |P|) \leq O(\sum_P C(P) \cdot M) < O(N \cdot M)$. As a result, the total time complexity for line 9-21 is $O(n \cdot N \cdot M)$. For line 22-25, the time complexity is $O(M \cdot n)$. For line 27-31, for every node in $R.set$, it needs to compute m values, and each value can be computed in time $O(N)$. The total time complexity for line 27-31 is $O(m \cdot N \cdot M)$. From the analysis, the total time complexity for Algorithm 5 is $O(M \cdot N + n \cdot N \cdot M + M \cdot n + m \cdot N \cdot M) = O((n+m) \cdot N \cdot M)$. □

Cost Sharing Cross LDs: In addition, when there are many LDs computed, it is costly to evaluate each LD individually. LDs from $\mathsf{LV}(R)$ and $\mathsf{LV}(R')$ may share subtrees. As shown in Fig. 8, when evaluating the two LVs, LV(Complaints) and LV(Orders), they share the LV(Customers) and LV(Products). Therefore, the information computed in the bottom-up phase for Customers, and Products can all be reused for computing the Gtrees for LV(Complaints) and LV(Orders). It can be done by adding the following before the start of the bottom-up phase: if $\mathsf{LV}(P)$ has been evaluated before, skip it and continue for the next node. Here, P is a relation node in $\mathsf{LD}(R)$ which represents a subtree rooted at P in $\mathsf{LD}(R)$.

VI. Related Work

Keyword Based Aggregation on Databases: Wu et al. [27] studied the keyword-driven OLAP (KDAP). First, it finds and ranks the subspaces that can best describe a set of keywords by considering several ambiguities for keywords. Second, the user chooses the best subspace generated in the first phase, and partitions the chosen subspace into dynamic facets that allow the user to roll-up and drill-down the result spaces. The aggregation involved in KDAP is based on some pre-defined measures that are independent of keyword relevance. Tata et al. in [26] integrated the aggregate function and the numerical attribute to be aggregated into a keyword query.

They focus on translating a user query into a set of proper interpretations where each must contain a numerical attribute and an aggregate function for aggregation.

Bhide et al. in [4] studied keyword search over dynamic categorized information. Given a set of keywords, [4] developed the CS* system to find the top-k categories that are most relevant to the user query. TF-IDF ranking issues are discussed in [4] to rank categories. In [4], categories have only one dimension and are all predefined based on the set of documents to be searched. Zhou et al. in [29] studied keyword based aggregation using minimal group-bys. Given a set of keywords, it returns all minimal subspaces (aggregate cells) that contain all the user given keywords. Tao and Yu in [25] proposed algorithms to find frequent co-occurring terms in relational keyword search. It finds the top-k terms that are always co-occurring with a set of user given keywords in the result of a traditional keyword search over *RDB*.

Multidimensional Search on Text Databases: Mothe et al. [21] proposed a user interface to provide users a global visualization of a large document collection. The interface enables OLAP and multidimensional analysis operators, where each dimension is represented by a concept hierarchy. Inokuchi et al. [13] proposed a data representation and algebra operations to analyze large sets of textual documents with metadata. Simitsis et al. [24] proposed a Multidimensional Content eXploration (MCX) system, which is to effectively analyze and explore large amount of content by combining keyword search with OLAP-style aggregation. The data considered are generally documents with limited metadata. The link structures of documents are taken into consideration to score documents in [24]. A Text-Cube model was proposed by Lin et al. [17] to study effective OLAP over text databases. Topic Cube was proposed by Zhang et al. [28] to combine OLAP with probabilistic topic modeling and enable OLAP on the dimension of general text data. In the multidimensional text database, a topic dimension is defined besides the traditional structured attributes dimensions, and topic hierarchy is defined to enable roll-up and drill-down operations. The topic model is based on probabilistic latent semantic analysis. The measures for cube cells are word distributions. It is worth noting that, for all of these works, the dimensions are systematically defined, and no keyword is used to specify the user-wanted dimensions.

Keyword Search on Relational Databases: In the literature, for a keyword query on a relational database, it returns a set of inter-connected structures in the *RDB* that contain the user given keywords. The techniques to answer keyword queries in *RDB*s are mainly in two categories: *CN*-based (schema-based) and graph based (schema-free) approaches.

In the *CN*-based approaches [1], [12], [10], [19], [20], it processes a keyword query in two steps, namely, candidate network (*CN*) generation and *CN* evaluation. *DBXplorer* [1] and DISCOVER [12] focused on retrieving connected trees using SQL on RDBMSs. In [20], Markowetz et al. discussed how to efficiently generate all *CN*s and how to process keyword queries in an *RDB* stream environment based on a sliding window model. Among these approaches, in *DISCOVER-II* [10], Hristidis et al. incorporated IR-style ranking techniques to rank the connected trees. In *SPARK* [19], Luo et al. proposed a new ranking function by treating each connected tree as a virtual document, and unified the AND/OR semantics in the score function using a parameter. The ranking issues were also discussed in [2], [11], [18].

Finding top-k interconnected structures has been extensively studied in graph based approaches in which an *RDB* is materialized as a weighted database graph $G_D(V, E)$. The representative works on finding top-k connected trees are [3], [14], [15], [6], [8]. In brief, finding the exact top-k connected-trees is an instance of the group Steiner tree problem [7], which is NP-hard. To find top-k connected trees, Bhalotia et al. proposed backward search in *BANKS-I* [3], and Kacholia et al. proposed bidirectional search in *BANKS-II* [14]. Kimelfeld et al. [15] proposed a general framework to retrieval top-k connected trees with polynomial delay under data complexity, which is independent of the underline minimum Steiner tree algorithm. Ding et al. in [6] also introduced a dynamic programming approach to find the minimum connected tree and approximate top-k connected trees. Golenberg et al. in [8] attempted to find an approximate result in polynomial time under the query and data complexity.

Top-k connected trees are hard to compute, in *BLINKS* [9], He et al. proposed the distinct root semantics. In *BLINKS*, search strategies were proposed with a bi-level index built to fast compute the shortest distances. Dalvi et al. [5] conducted keyword search on external memory graphs under the distinct root semantics. Li et al. in *EASE* [16] defined an r-radius Steiner graph, where each r-radius Steiner graph is a subpart of a maximal r-radius subgraph. Qin et al. studied multi-center communities under the distinct core semantics in [23], and proposed new polynomial delay algorithms to compute all or top-k communities.

It is worth noting that all the above works focus on finding individual keyword search results and no aggregation issues are involved.

VII. Performance Studies

We conducted extensive performance studies to test the algorithms proposed in this paper. We implemented two algorithms, denoted Naive and New. Both Naive and New algorithms follow the framework shown in Algorithm 1, and generate all LDs using Algorithm 3. For the Naive algorithm, we evaluate LDs using the naive approach shown in Algorithm 4. For the New algorithm, we evaluate LDs using Algorithm 5. Techniques for sharing cost across LDs are allowed in both the Naive algorithm and the New algorithm.

We use two large real datasets, *DBLP* (http://www.informatik.uni-trier.de/~ley/db/), and *IMDB* (http://www.imdb.com/interfaces) and a large synthetic dataset *TPC-H* (http://www.tpc.org/tpch/) for testing.

All algorithms were implemented in Visual C++ 2008 and all tests were conducted on a 2.8GHz CPU and 2GB memory PC running Windows XP. The time to construct the Dimensional Inverted Index for *DBLP*, *IMDB* and *TPC-H* are

Key Freq	Keywords
1	flexible resource pattern
2	communication optimal implementation
3	application knowledge dynamic programming method
4	parallel web performance
5	information data system

Fig. 9. Keywords used for the *DBLP* dataset

Key Freq	Keywords
1	radio dance place
2	black Kevin manager
3	part Smith driver party music
4	animation captain Martin
5	Jack Tom Paul

Fig. 10. Keywords used for the *IMDB* dataset

4.5 minutes, 12.4 minutes and 16.9 minutes respectively. We tested several queries with different keyword frequencies. We use the aggregation function sum as β, and the TF-IDF score function α introduced in [19] in the experiments .

For *DBLP*, the schema includes the following four relations: Papers(Paperid, Conference, Year, Title), Authors(Authorid, Authorname, Position, Affiliation, University, Interest), Write(Writeid, Authorid, Paperid) and Cite(Citeid, Paperid1, Paperid2). The primary key for each relation is underlined. The size of the raw data for the *DBLP* dataset is 643MB. The number of tuples for the four relations are 2,045,150, 758,839, 4,033,272 and 112,435 respectively, and the total number of tuples in *DBLP* is 6,949,696.

The *IMDB* schema includes 8 relations: Actors(Actorid, Gender, Actorname), Act(Actid, Actorid, Movieid, Charactor), Directors (Directorid, Directorname), Direct (Directid, Directorid, Movieid), Movies(Movieid, Genreid, Languageid, Locationid, Year, Name), Genres(Genreid, Genre), Languages(Languageid, Language) and Locations(Locationid, Location, Area, City, State, Country). The primary key for each relation is underlined. The size of the raw data for the *IMDB* dataset is 920MB. The number of tuples in the eight relations are 1,774,431, 12,514,476, 180,702, 1,026,444, 1,501,623, 28, 315 and 63,382 respectively, and the total number of tuples in *IMDB* is 17,061,401.

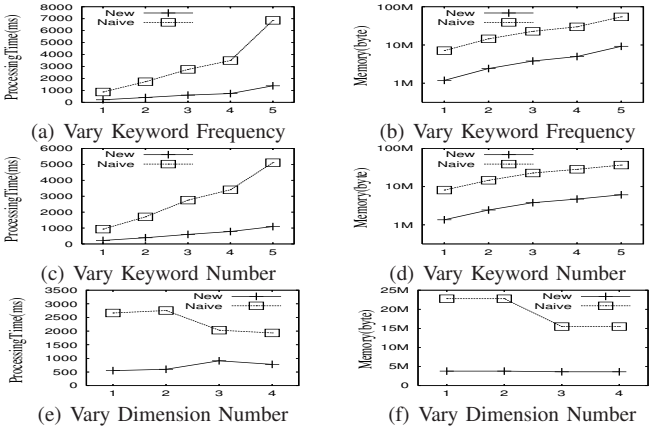
For *TPC-H*, the schema includes the following eight relations: Part, Supplier, Partsupp, Customer, Nation, Lineitem, Region, Orders. There are totally 62 attributes in the eight relations. The size of the raw data for the *TPC-H* dataset is 1,005MB. The number of tuples for the 8 relations are 200,000, 10,000, 800,000, 150,000, 25, 6,001,215, 5 and 1,500,000 respectively, and the total number of tuples in *TPC-H* is 8,661,245.

For the scalability testings, we report the processing time and memory consumption for each test case. The processing time includes the time for generating and evaluating all LDs for the query. For each dataset, we select representative queries with different keyword frequencies as follows. After removing all the stop words, we set the maximum keyword frequency among all keywords as τ, and divide the keyword frequency range between 0 and τ into 5 partitions, namely, $\tau/5$, $2\tau/5$, $3\tau/5$, $4\tau/5$ and τ. For simplicity, we say a keyword has frequency p ($p \in \{1, 2, 3, 4, 5\}$), if and only if its frequency is between $(p-1) \cdot \tau/5$ and $p \cdot \tau/5$.

For all scalability testings, we vary 3 parameters, namely, the keyword frequency, the keyword number, and the dimension number. The keyword frequency is the frequency for each general-keyword in the query. The keyword number is the number of general-keywords used in a query, and the dimension number is the number of dimensional-keywords used in a query. Every parameter has a default value. For *DBLP*, the settings are similar to *IMDB*. The general-keywords selected with different keyword frequencies are shown in Table 9 and by default, the keyword frequency is 3. The keyword number ranges from 1 to 5 with a default value 3 and the dimension number ranges from 1 to 4 with a default value 2. The set of dimensional-keywords are selected from {"conference", "year", "authorname", "20th"} and by default it is {"conference", "year"}. For *IMDB*, the keyword frequency ranges from 1 to 5 with a default value 3. The general-keywords selected with different keyword frequencies are shown in Table 10. The keyword number ranges from 1 to 5 with a default value 3, and the dimension number ranges from 1 to 5 with a default value 3. When varying keyword number k, we select the first k keywords from the 5 keywords shown in the third line of Table 10. The dimensional-keywords from all testings in *IMDB* are selected from {"year", "genre", "USA", "male", "city"} and by default it is {"year", "genre", "USA"}. When varying the dimension number d, we select the first d keywords from the 5 dimensional-keywords. For *TPC-H*, the settings are all similar to *IMDB* and *DBLP*.

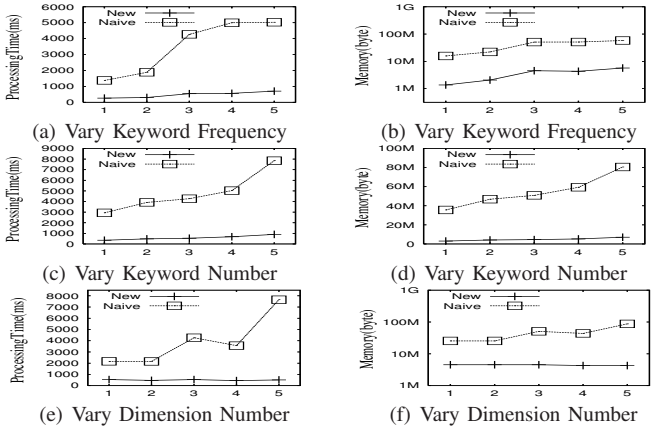
Exp-1 (Effectiveness Testing): In order to demonstrate the effectiveness of our approach, in addition to the six examples demonstrated in the introduction part, we show and analyze another two representative queries, Qa and Qb tested against *DBLP*, and another three representative queries, Qc, Qd, and Qe, tested against *IMDB*. In each query, the dimensional-keywords and general-keywords are separated by a semicolon. (**Qa**) {"conference", "year"; "relational", "database", "management"} asks "in which conference and year, there are most papers about relational database management" in *DBLP*. We obtained the following result. Under the LD consisting of edges (Papers $\rightarrow$ conference, Papers $\rightarrow$ year), the groups with highest scores are (SIGMOD, 2005, 41.8), (SIGMOD, 1982, 33.8), and (SIGMOD, 1979, 27.2). Under the LD consisting of edges (Write $\rightarrow$ Papers, Papers $\rightarrow$ conference, Papers $\rightarrow$ year), the groups with highest scores are (TODS, 1976, 199.8), (SIGMOD, 2005, 96.7), and (VLDB, 2005, 95.4). After searching the *DBLP* website, we found that SIGMOD'05, SIGMOD'82, SIGMOD'79, TODS'76 and VLDB'05 all have many papers about "relational database management". Under the second LD, TODS'76 has the highest score because the paper "System R: Relational Approach to Database Management" in TODS'76 has as many as 14 authors.

(**Qb**) {"authorname", "2000"; "subgraph" "isomorphism"} asks "which author in the year 2000 has most papers about subgraph isomorphism" in *DBLP*. We obtained the following result. Under the LD consisting of edges (Write $\rightarrow$ Authors, Authors $\rightarrow$ authorname, Write $\rightarrow$ Papers, Papers $\rightarrow$ year), the groups with highest scores are (Ruth V. Spriggs, 2000, 50.0), (Shuichi Ichikawa, 2000, 33.3), and (Lerdtanaseangtham



(a) Vary Keyword Frequency (b) Vary Keyword Frequency
(c) Vary Keyword Number (d) Vary Keyword Number
(e) Vary Dimension Number (f) Vary Dimension Number

Fig. 11. Testing Results for the *DBLP* Dataset



(a) Vary Keyword Frequency (b) Vary Keyword Frequency
(c) Vary Keyword Number (d) Vary Keyword Number
(e) Vary Dimension Number (f) Vary Dimension Number

Fig. 12. Testing Results for the *IMDB* Dataset

Udorn, 2000, 33.3). After searching the *DBLP* website, we found that all of the three authors have more than 2 papers about subgraph isomorphism in the year 2000.

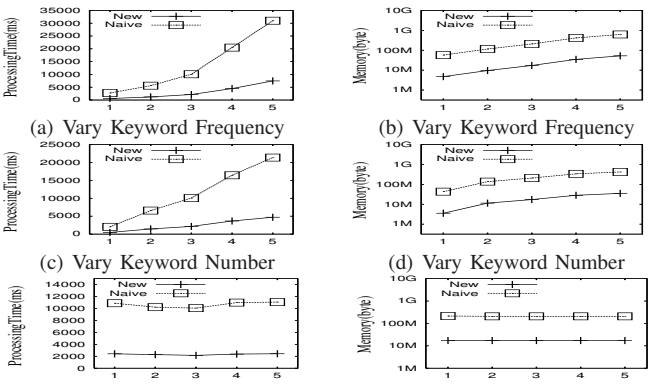
(**Qc**) {"city", "USA"; "musical"} asks "for which city in USA, there are most musical movies" in *IMDB*. We obtained the following result. Under the LD consisting of edges (Movies → Locations, Locations → city, Locations → country), the groups with highest scores are (New York City, USA, 528.5), (Los Angeles, USA, 492.0), (Culver City, USA, 68.1), and (Las Vegas, USA, 62.4). It indicates that most musical movies in USA are from the New York City and Los Angeles. Under the LD consisting of edges (Act → Movies, Movies → Locations, Locations → city, Locations → country), the groups with highest scores are (New York City, USA, 11691.3), (Los Angeles, USA, 9998.2), (Culver City, USA, 2390.0), and (Big Bear Lake, USA, 1562.1). Here, the group (Big Bear Lake, USA) has a higher rank because movies in Big Bear Lake have more actors than movies in Las Vegas.

(**Qd**) {"language", "gender", "genre"; "sing", "song"} asks "for which language, gender, and genre, there are most movies about sing and song" in *IMDB*. We obtained the following results. Under the LD consisting of edges (Act → Actors, Actors → gender, Act → Movies, Movies → Genres, Genres → genre, Movies → Languages, Languages → language), the groups with highest scores are (English, male, musical, 1421.1), (English, male, comedy, 1186.9).

(**Qe**) {"action", "actorname"; "spider"} asks "which actor acts most action movies about spider" in *IMDB*. We obtained the following result. Under the LD consisting edges (Act → Actors, Actors → actorname, Act → Movies, Movies → Genres, Genres → genre), the groups with highest scores are (action, Tobey Maguire, 79.9), (action, Bruce Campbell, 37.4), and (action, Kirsten Dunst, 37.4). The actor Tobey Maguire has the highest score because he is an actor in the movies from "Spider-Man 1" to "Spider-Man 5" and he himself acts the spider-man in the movies. The other persons are all the actors for the "Spider-Man" movies.

Exp-2 (Scalability Testing for *DBLP*): Fig. 11(a) and Fig. 11(b) show that, when the keyword frequency increases in *DBLP*, the time/memory consumption for both Naive and New algorithms increases. The New algorithm consumes 6 times less CPU and 10 times less memory than the Naive algorithm. In Fig. 11(c) and Fig. 11(d), when the number of general-keywords increases, the New algorithm also performs much better than the Naive algorithm. When increasing the dimension number, as shown in Fig. 11(e) and Fig. 11(f), the time/memory consumption for the New algorithm is consistent, but the time/memory consumption for the Naive algorithm has a trend to decrease. This is because for this query, after adding a certain dimensional-keyword (e.g., authorname), some of the previously generated LDs does not contain any attribute that can match the new dimensional-keyword. As a result, the total number of LDs generated decreases, which causes the time/memory consumption decreasing for the Naive algorithm. The New algorithm also performs much better than the Naive algorithm in all cases.

Exp-3 (Scalability Testing for *IMDB*): The testing results for *IMDB* are shown in Fig. 12. From Fig. 12(a) and Fig. 12(b), we know that when the keyword frequency increases, the processing time and memory consumption for both Naive and New algorithms increase. This is because when keyword frequency increases, the probability for each tuple to contain a keyword increases. As a result, the number of nodes participated in both Naive and New algorithms increases, which increases the time/memory consumption for both algorithms. The Naive algorithm consumes 5 times more processing time and 10 times more memory than the New algorithm in all cases, because the New algirthm avoids computing much useless/redundancy information using the two-phase approach than the naive algorithm. Fig. 12(c) and Fig. 12(d) show that, when the number of keywords increases, the processing time and memory consumption for both Naive and New algorithms increase. The New algorithm also performs much better than the Naive algorithm for both time and memory consumption. In Fig. 12(e) and Fig. 12(f), we increase the dimension number by adding another dimensional-keyword each time. The processing time/memory consumption for the Naive algorithm has a trend to increase, because in this case, when the dimension number increases, more LDs will be generated because some of the dimensional-keywords such as "USA" and "city" can be matched to many attributes in the relations. But after adding the dimensional-keyword "male",



(a) Vary Keyword Frequency (b) Vary Keyword Frequency
(c) Vary Keyword Number (d) Vary Keyword Number
(e) Vary Dimension Number (f) Vary Dimension Number

Fig. 13. Testing Results for the *TPC-H* Dataset

the processing time/memory consumption decreases because the constraint "male" on the gender attribute can decrease the number of tuples matched in the final result, thus the time for aggregating and score computing decreases. The performance of the New algorithm is also much better than the Naive algorithm and is consistent when the dimension number increases, because much computational cost is shared when evaluating LDs.

Exp-4 (Scalability Testing for *TPC-H*): The testing results for *TPC-H* are shown in Fig. 13. From Fig. 13(a) to Fig. 13(d), the curves for the *TPC-H* dataset for both algorithms are all similar to the curves in the *DBLP* and *IMDB* datasets when varying the keyword frequencies and keyword numbers. In Fig. 13(e) and Fig. 13(f), when the number of dimensional-keywords increases, the performance for both algorithms keeps consistent. This is because when the number of dimensional-keywords increase, the number of dimensions for each result increases, and thus the cost for computing such dimensions increases. On the other hand, the number of LDs generated decreases, because some of the LDs will not match the new added dimensional-keyword, and thus the cost for computing groups in all LDs decreases.

VIII. Conclusion

In this paper, we studied a new keyword query, $Q = (Q_d, Q_g, \alpha, \beta)$, to compute structural statistics for all groups of tuple-trees in an *RDB*. We represent the *RDB* as a data graph G_D. We show how to compute Dtrees, as m-dimensional objects, that must contain all the keywords in Q_d, for $m = |Q_d|$, over G_D. We also show how to compute Gtrees, that contain keywords in Q_g and are strongly related to an m-dimensional object. We discuss how to group m-dimensional objects into groups without tree isomorphism testing, using the label-trees (LDs) based on the database schema. We present a two-step approach, with a new algorithm to compute all label-trees for a structural statistics keyword query, and a new two-phase algorithm to compute group-&-aggregate using the label-trees computed. We discussed cost sharing, when computing a single label-tree, and cost sharing, when computing cross several label-trees. We analyze our algorithm and provide time complexity. We conducted extensive experimental studies using two large real datasets, *IMDB* and *DBLP* and a synthetic dataset *TPC-H*, and confirmed the effectiveness and efficiency of our approach.

Acknowledgment: The work was supported by grants of the Research Grants Council of the Hong Kong SAR, China No. 419008 and 419109.

References

[1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *Proc. of ICDE'02*, 2002.
[2] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *Proc. of VLDB'04*, 2004.
[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *Proc. of ICDE'02*, 2002.
[4] M. Bhide, V. T. Chakaravarthy, K. Ramamritham, and P. Roy. Keyword search over dynamic categorized information. In *Proc. of ICDE'09*, pages 258–269, 2009.
[5] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *PVLDB*, 1(1), 2008.
[6] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *Proc. of ICDE'07*, 2007.
[7] S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. In *Networks*, 1972.
[8] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *Proc. of SIGMOD'08*, 2008.
[9] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *Proc. of SIGMOD'07*, 2007.
[10] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-Style keyword search over relational databases. In *Proc. of VLDB'03*, 2003.
[11] V. Hristidis, H. Hwang, and Y. Papakonstantinou. Authority-based keyword search in databases. *ACM Trans. Database Syst.*, 33(1), 2008.
[12] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proc. of VLDB'02*, 2002.
[13] A. Inokuchi and K. Takeda. A method for online analytical processing of text data. In *Proc. of CIKM'07*, pages 455–464, 2007.
[14] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proc. of VLDB'05*, 2005.
[15] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *Proc. of PODS'06*, 2006.
[16] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *Proc. of SIGMOD'08*, 2008.
[17] C. X. Lin, B. Ding, J. Han, F. Zhu, and B. Zhao. Text cube: Computing ir measures for multidimensional text database analysis. In *Proc. of ICDM'08*, pages 905–910, 2008.
[18] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *Proc. of SIGMOD'06*, 2006.
[19] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *Proc. of SIGMOD'07*, 2007.
[20] A. Markowetz, Y. Yang, and D. Papadias. Keyword search on relational data streams. In *Proc. of SIGMOD'07*, 2007.
[21] J. Mothe, C. Chrisment, B. Dousset, and J. Alau. Doccube: Multi-dimensional visualisation and exploration of large document sets. *JASIST*, 54(7):650–659, 2003.
[22] L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: The power of rdbms. In *Proc. of SIGMOD'09*, 2009.
[23] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *Proc. of ICDE'09*, 2009.
[24] A. Simitsis, A. Baid, Y. Sismanis, and B. Reinwald. Multidimensional content exploration. *PVLDB*, 1(1):660–671, 2008.
[25] Y. Tao and J. X. Yu. Finding frequent co-occurring terms in relational keyword search. In *Proc. of EDBT'09*, pages 839–850, 2009.
[26] S. Tata and G. M. Lohman. Sqak: doing more with keywords. In *Proc. of SIGMOD'08*, pages 889–902, 2008.
[27] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *Proc. of SIGMOD'07*, pages 617–628, 2007.
[28] D. Zhang, C. Zhai, and J. Han. Topic cube: Topic modeling for olap on multidimensional text databases. In *Proc. of SDM'09*, pages 1123–1134, 2009.
[29] B. Zhou and J. Pei. Answering aggregate keyword queries on relational databases using minimal group-bys. In *Proc. of EDBT'09*, pages 108–119, 2009.