OSIR 2006

Second Workshop On

Open Source Information Retrieval

In Conjunction with the 2006 ACM SIGIR Conference
August 10, 2006
Seattle, WA

Edited by Michel Beigbeder, Wray Buntine and Wai Gen Yee

OSIR 2006 Organization

Workshop chairs

Michel Beigbeder
G2I Department
École Nationale Supérieure des Mines de Saint-Étienne, France

Wray Buntine
Complex Systems Computation Group
Helsinki Institute for Information Technology, Finland

Wai Gen Yee
Department of Computer Science
Illinois Institute of Technology, USA

Program Committee

Antonio Badia, University of Louisville, USA
Michel Beigbeder, École Nationale Supérieure des Mines de Saint-Étienne, France
Wray Buntine, Helsinki Institute for Information Technology, Finland
Abdur Chowdhury, America Online, USA
Bruce Croft, University of Massachusetts, USA
Doug Cutting, Yahoo, USA
Ophir Frieder, Illinois Institute of Technology, USA
Nazli Goharian, Illinois Institute of Technology, USA
Donald Kraft, Louisianna State University, USA
Olfa Nasraoui, University of Louisville, USA
Iadh Ounis, University of Glasgow, UK
Michael Stack, Internet Archive, USA
Wai Gen Yee, Illinois Institute of Technology, USA
Clement Yu, University of Illinois at Chicago, USA

Workshop Web site

www.emse.fr/OSIR06

OSIR 2005

Open Source Information Retrieval

The World Wide Web has grown to be a primary source of information for millions of people. Due to the size of the Web, search engines have become the major access point for this information. However, "commercial" search engines use hidden algorithms that put the integrity of their results in doubt, collect user data that raises privacy concerns, and target the general public thus fail to serve the needs of specific search users. Open source search, like open source operating systems, offers alternatives.

The goal of the Open Source Information Retrieval Workshop (OSIR) is to bring together practitioners developing open source search technologies in the context of a premier IR research conference to share their recent advances, and to coordinate their strategy and research plans. The intent is to foster community-based development, to promote distribution of transparent Web search tools, and to strengthen the interaction with the research community in IR.

A workshop about Open Source Web Information Retrieval was held last year in Compigne, France as part of WI 2005. The focus of this worksop is broadened to the whole open source information retrieval community.

We want to thank all the authors of the submitted papers, the members of the program committee: Antonio Badia, Abdur Chowdhury, Bruce Croft, Doug Cutting, Ophir Frieder, Nazli Goharian, Donald Kraft, Olfa Nasraoui, Iadh Ounis, Michael Stack, Clement Yu, and the several reviewers whose contributions have resulted in these high quality proceedings.

Michel Beigbeder, Wray Buntine and Wai Gen Yee

Table of contents

Low Latency Index Maintenance in Indri

Trevor Strohman
strohman@cs.umass.edu

W. Bruce Croft
croft@cs.umass.edu

Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts
Amherst, MA 01003

ABSTRACT

There has been a resurgence of interest in index maintenance (or incremental indexing) in the academic community in the last three years. Most of this work focuses on how to build indexes as quickly as possible, given the need to run queries during the build process.

This work is based on a different set of assumptions than previous work. First, we focus on latency instead of throughput. We focus on reducing index latency (the amount of time between when a new document is available to be indexed and when it is available to be queried) and query latency (the amount of time that an incoming query must wait because of index processing). Additionally, we assume that users are unwilling to tune parameters to make the system more efficient.

We show how this set of assumptions has driven the development of the Indri index maintenance strategy, and describe the details of our implementation.

1. INTRODUCTION

One of the biggest successes of information retrieval research has been the development of efficient full-text indexing systems. The compressed inverted list index allows for quick retrieval of documents from very large corpora. Web search engines have put this technology to public use, and now hundreds of millions of people use full-text search every day.

Most research into text indexing assumes that the collection of documents to be searched is unchanging. However, it is difficult to imagine many useful scenarios where this is a reasonable assumption. Web documents, news feeds, e-mails, corporate documents, and even library book collections are constantly changing. Even if the documents themselves are static, most collections of documents grow over time, and the newest documents are often the most relevant. Handling this changing nature of data is an important area of information retrieval research.

The process of updating information in an existing index is called both index maintenance and incremental indexing in the literature. This is a small but active field of research. However, the evaluation metrics used in most papers is on the throughput of the indexing process. We claim that indexing throughput is not the critical factor in many real scenarios. Modern information retrieval systems can handle as much as 50GB per hour of new text data on a single machine [4], and other than the web, few data sources generate new or changed text at that rate.

By contrast, consider a search engine storing data from a particular newswire. The TREC[1] AP89 collection consists of 84,678, for an average of 261 new news stories a day. This collection is small by current standards, and the rate of information change cannot be considered taxing for any modern system. Raw indexing throughput is not the most important measure for this application. We argue that latency is a better measure–the amount of time taken between the appearance of a document on the wire until it is able to be searched, and the amount of time that incoming queries must wait while the indexing process for a new document is taking place.

We have built an incremental indexing system that focuses on this latency problem. The remainder of this paper focuses on describing our system and relating it to previous work. We do not have an evaluation of our system at this time, but we hope that this paper will spark additional work in the future.

2. RELATED WORK

While there has always been practical commercial interest in incremental indexing, academic work on this topic has been sparse. A small pool of researchers considered this topic between 1990 and 1995, then publications nearly stopped. In the past three years, new research has arrived to build upon and update the assumptions in earlier work.

Cutting and Pedersen [7] present the first incremental indexing work that we consider here. The authors use a variety of methods to store posting lists so that they may be dynamically updated. In the first method, they store postings (word and document location pairs) directly in the B-Tree, sorted first by word and second by document location. This straightforward approach is used more recently by the MySQL database engine for full-text search [1]. This

[1]http://trec.nist.gov

OSIR'06, August 11, 2006, Seattle, Washington, USA.
ACM .

Disk Model	Transfer rate	Seek time	Year	Size
Seagate ST1980	3.55 MB/s	9.9 ms	1994	1GB
Seagate ST3146854FC	80 MB/s	3.5 ms	2004	147GB

Table 1: Performance figures for two Seagate hard disk models produced in the last decade.

method is simple and conceptually clean, but leaves room for improvement in both space and speed. The authors improve on space utilization by storing the word only once, instead of in each posting. Then, they improve on speed by using an external heap file to store list data instead of storing the data within the tree itself.

Tomasic, Garcia-Molina and Shoens [14] focus on the storage allocation policy in the inverted file. As in Cutting and Pedersen's approach, the inverted file is a heap that requires an allocation policy. Clearly the individual lists are expected to grow over time, but leaving large gaps in the file for extra postings wastes space and time (since the extra file space implies longer seek times between relevant data regions). The authors explore three allocation policies: constant, block, and proportional. In the constant case, each inverted list update operation reserves a constant amount of extra space for new postings. The block strategy is similar, except the extended list is forced to end on a block boundary. The proportional strategy increases leaves some percentage of the total length of the list as empty space; this means that longer lists have more room to grow. Additionally, the authors consider three update policies; new, whole and fill. The new strategy writes new postings to a new location, effectively making each inverted list a linked list of segments. The fill strategy is similar, except each linked list segment is forced to be the same length. Finally, the whole strategy requires that each inverted list be copied at every update, so that lists remain contiguous. Not surprisingly, the authors find that the *new* strategy is quicker for updates, while the *whole* strategy is preferred for queries.

Brown, Callan and Croft [2] investigate similar approaches to those of Tomasic et al. [14]. The authors modify the INQUERY retrieval system to store its data in the Mneme object store [11]. This abstraction of the storage layer is similar to more recent work, such as de Vries et al. [8]. Brown et al. store inverted lists in this disk-based object store; small lists are segregated from large ones, and small lists are allocated in power-of-two sized blocks. Large lists are not necessarily stored sequentially, but may be stored in a linked list of blocks. The results are largely similar to Tomasic et al. [14].

Clarke et al. [5] present a system which, in contrast with the others shown so far, explicitly discusses query activity while new documents are added to the collection. Unlike the work shown in this paper, this method still adds documents to the index in batches, but these updates are committed quickly to the index, so that pauses in query operations are as short as possible.

The four papers mentioned here represented the state-of-the-art for index maintenance until recently. However, these papers were written in the mid-1990s; since that time, computer technology has changed drastically.

The disk models shown in Table 1 are a small sample of the disks available on the market, but they represent the progress that drive makers have achieved over the last decade. Seek times have dropped by approximately 60%, but transfer rates have increased by as much as 20 times. Just looking at these figures, it seems that disk seeks have become significantly more costly relative to disk transfers.

This comparison is not completely fair, because average seek time is computed as the average time to seek between two random locations on the device. Since the new devices are bigger, the 'average' seek time represents a jump over much more data. In the 1994 disk, for example, an average seek might cross 500MB of data, while on a newer disk it might cross 70GB of data. Therefore, we expect that if the same size document index was used on both drives, the seek time difference would be more dramatic.

A more subtle change is in the amount of time necessary to read the data of the entire disk. For the 1994 drive, the entire contents of the drive can be transferred in about 4 minutes. The 2004 drive requires 30 minutes for the same task. If we assume that collection sizes keep pace with disk sizes, we see that transfer rates are not keeping up with increasing data size.

More details on the changes in retrieval system performance can be found in Zobel, Williams and Kimberly [15]. The important factor is that research that relies on disk performance must be revisited in order to maintain relevance.

Lester et al. [10] revived interest in the area of index maintenance with a 2004 study on the relative advantages of three strategies, in-place, re-build and re-merge. The in-place strategy is similar to Tomasic's *whole* strategy with a proportional allocation policy. The re-build strategy simply rebuilds the entire collection, while re-merge merges new postings into an existing index, forming a new index. The authors find that for updates of less than 10000 documents, both incremental strategies are better than rebuilding the index. Furthermore, for the smallest updates (under 100 documents), the in-place strategy is faster. However, for these small updates, the update time per document approaches 1 second.

Especially because of the issues in transfer time, a new class of update algorithms has been discussed in the literature recently. This strategy, called *geometric partitioning* by Lester et al. [9], does not force inverted lists to be merged together when postings are flushed to disk. Instead, new postings are flushed to disk in entirely new indexes, called *partitions*. Queries acting upon this data must check each partition for inverted list data, so there is a query speed penalty for maintaining too many partitions. Therefore, these partitions can be merged together to form larger partitions that are more efficient to query. Since merging is costly, an efficient system must balance the needs of query processing and efficient indexing in choosing its merging policy.

The name geometric partitioning refers specifically to a merging policy developed by Lester et al. [9]. For a system that can hold b document pointers in memory and some positive integer parameter r, the i^{th} partition is limited in size to br^i pointers. For example, if $r = 3$, the first disk partition is limited to holding 3 times as many documents as the system memory can hold; if this partition grows beyond that size, its data is merged into the second partition (which is limited to 9 times the system memory). By keeping this exponential distribution of partition sizes, the total number of indexes is kept small while still making the common case (merging in-memory data into the first partition) fast.

Many authors ([3], [4], [9], [12]) have studied the parti-

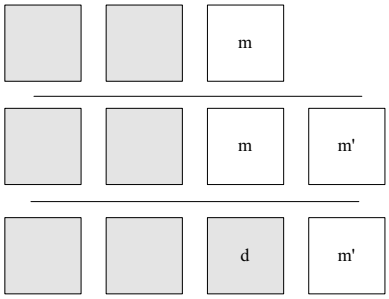


Figure 1: Three steps in the process of adding a new memory partition. First, a new partition m' is added. Next, m is written to disk asynchronously. Next, d is made available for new queries. Finally, when m is no longer in use by any query, it is deleted.

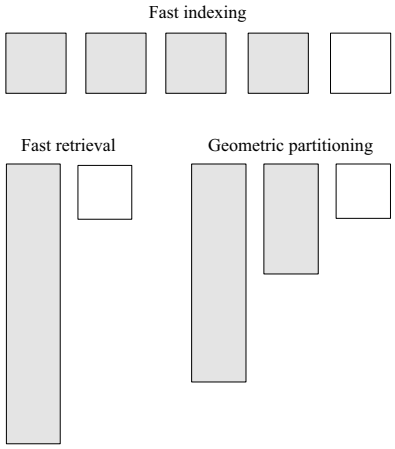


Figure 2: Three possible partition merging policies. The 'fast indexing' policy avoids merging partitions as much as possible; this method approximates a traditional batch indexer. The 'fast retrieval' policy eagerly merges new data, making a single large disk partition. The 'geometric partitioning' strategy finds a balance between the other two policies.

tioning strategy with good results. However, previous work has focused on throughput instead of latency of operations. In this work, we consider latency as our first goal.

3. IMPLEMENTATION DETAILS

3.1 Overview

Our implementation is a part of the Indri search engine [13]. The implementation described here has been in the toolkit since the 2.0 release.

Indri is a component of the Lemur toolkit for Information Retrieval [6]. The Lemur toolkit is an open-source system built to enable Information Retrieval research using language modeling. Indri extends upon the goals of the Lemur toolkit by adding new structured document retrieval functionality and a flexible query language. In addition, Indri is built to be used by both researchers and industrial users.

One common use of Indri is to index news feed data. Like most search engines on modern hardware, it is not difficult for Indri to keep up with the volume of data generated by a news feed. However, it is desirable that new news articles be made available immediately. Furthermore, the addition of a new news article should not be able to halt query processing for a noticeable period of time. This set of requirements drove our implementation of a concurrent version of Indri.

3.2 Index design

As in the work of Lester et al. [9], Indri uses index partitions to maintain high query and indexing throughput. In many previous systems, data stored in memory was not considered fully indexed, and data needed to be flushed to disk in order to be queried. This is not the case in Indri; the first partition is stored in RAM, and can be queried directly. This allows Indri to support high-speed indexing and query processing for smaller collections without ever requiring disk seek delays.

For collections that grow larger than memory can hold, Indri writes partitions to disk. Once a partition has been written to disk, the data will never be changed–it may be merged into another partition and then deleted, but never changed. This allows disk data to be read without the need for locks.

In memory, term statistics and vocabulary are stored in a hash table. On disk, vocabulary information is stored in two B-Trees, with one dedicated to frequent terms and the other dedicated to infrequent terms. The frequent terms tree (consisting of terms that appear more than 1000 times in the collection) is small enough that it always fits easily in memory. The infrequent terms tree is larger than system memory for large collections. The inverted list information for all terms is stored in a single file, sorted by alphabetical order of term. The postings in each list are stored in document order.

To keep index latency as low as possible, we do not let disk I/O block indexing operations. All new documents are added to the in-memory partition. A monitoring thread checks periodically to see if the in-memory partition m has grown too large. If so, a new memory partition m' is created to accept new documents (Figure 1). Then, m is written to disk in a background thread. While m is being written to disk, it remains available for query processing. When the disk write completes, m has an on-disk representation, d, which will be used in place of m for all future queries. However, m may still be in use by some long-running queries. When all of those queries have completed, m is safely deleted.

In other systems, it is common for the parsing and indexing functions to be intertwined. Each word is parsed from the document, then added immediately to the index. In Indri, adding documents to the memory index requires a lock to prevent a query processing thread from seeing inconsistent data. Therefore, to keep index lock duration down, Indri parses each document completely (while not holding index locks) before adding document data to the index.

3.3 Query processing

Query processing is made more complicated by the existence of many index partitions. In Lester et al. [9], in-

verted lists for each query term are materialized in memory by copying data from each fragment into memory, then processing the combined list. Indri works differently; each partition is treated as a separate index. In the first phase of query processing, term statistics (such as term frequency) are collected from all partitions. In the second phase, the query is processed separately on each index partition, starting with the oldest data. When all partitions have been visited, query processing completes.

Since all data on disk is immutable, no special locking is necessary to access it. Indri allows many queries to execute simultaneously on this read-only data. When query processing reaches the most recent data (in the in-memory partition), the query processor must acquire a lock to ensure that the memory partition does not change while query processing occurs. The query processor acquires a read lock, so that other queries can operate on the in-memory data simultaneously. While this read lock does cause an increase in index latency, this latency can be reduced arbitrarily by decreasing the size of the in-memory partition.

Since merging happens asynchronously (as described in the indexing section), long-running queries can continue to execute while merging is in progress. The only point of contention between indexing and querying operations happens in the in-memory partition, where I/O operations are never performed while a lock is held.

3.4 Merging policy

In other recent index maintenance work, authors have proposed particular fixed merging policies. Parameters can be tuned in these policies to produce higher query throughput or higher indexing throughput.

However, in the kind of cases we are considering, we expect the optimal policy to change over time. In a desktop search application, we may find the system flooded with new data as a user downloads a large batch of documents onto the system. When this large batch of documents arrives, it is advantageous to tune the system for optimal indexing performance. However, later in the day, a user may start running queries against the data (while no new data is entering the system). At this point, optimum query performance would be preferred.

Indri attempts to pick a merging policy based on the weighted load average of the system over the previous 15 minutes (Figure 2). Every query is counted, as is every added document. When no queries have been executed in 15 minutes, the system is optimized for document indexing, and the system only merges index partitions when it is close to running out of file handles. Similarly, when no documents have been added for 15 minutes, the system chooses to merge data as often as possible in order to make the index structures optimized for query processing. Load between these two extremes causes the system to find a balance between query speed and indexing speed.

Our performance tuning so far has caused us to choose to merge partitions when:

- More than one partition exists, and
- The system has not thrashed in the last 5 minutes (that is, has not had to halt document additions because the system could not write them to disk quickly enough), and:
  - either:
    $$\frac{\text{documents added per second}}{50 \times \text{queries per second}} < |\text{partitions to merge}|$$
  - or there is very high query load, or
  - or there are very few documents being added.

This formulation is clearly *ad hoc*, and we have not performed a detailed analysis to determine how close to optimal this policy is. We do know that it seems to work well for the kinds of tasks we have tried. It works well enough that we have not seen the need to expose these parameters to the user.

4. CONCLUSION

While most work in index maintenance has focused on maximizing throughput, we considered the problem of reducing latency in a concurrent information retrieval system. Our system, Indri, contains a concurrent indexing and retrieval system built to reduce query and indexing latency, while making intelligent decisions about merging based on workload.

Our goal with this work is to stimulate new thinking about workloads and applications for index maintenance, particularly in evaluation. In the future, we hope to evaluate our system merging policy and latency in order to find ways that our implementation can be improved.

5. ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and in part by NSF grant #CNS-0454018 . Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the sponsor.

6. REFERENCES

[1] MySQL. `http://www.mysql.com/`.

[2] E. Brown, J. Callan, and W. Croft. Fast incremental indexing for full-text information retrieval. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB)*, pages 192 – 202, Santiago, Chille, September 1994.

[3] S. Büttcher and C. L. A. Clarke. Indexing time vs. query time trade-offs in dynamic information retrieval systems. In *CIKM 2005: Proceedings of the 14th ACM Conference on Infromation and Knowledge Management*, Bremen, Germany, Nov. 2005.

[4] S. Büttcher and C. L. A. Clarke. A hybrid approach to index maintenance in dynamic text retrieval systems. In *ECIR 2006: Proceedings of the 28th European Conference on Information Retrieval*, London, UK, Apr. 2006.

[5] C. L. A. Clarke, G. V. Cormack, and F. J. Burkowski. Fast inverted indexes with on-line update. Technical Report CS-94-40, University of Waterloo, Waterloo, Canada, 1994.

[6] W. B. Croft, J. Callan, J. Allan, C. Zhai, D. Fisher, T. T. Avrahami, T. Strohman, D. Metzler, P. Ogilvie, M. Hoy, J. Lafferty, J. Brown, L. Si, K. Collins-Thompson, M. Bilotti, F. Feng, and L. Larkey. The Lemur Project. `http://www.lemurproject.org/`.

[7] D. Cutting and J. Pedersen. Optimizations for dynamic inverted index maintenance. In *Proceedings of the 13th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 405–411, 1990.

[8] A. de Vries, J. List, and H. Blok. The multi-model DBMS architecture and XML information retrieval. *Intelligent Search on XML, Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, pages 179–192, 2003.

[9] N. Lester, A. Moffat, and J. Zobel. Fast on-line index construction by geometric partitioning. In A. Chowdhury, N. Fuhr, M. Ronthaler, H.-J. Schek, and W. Teiken, editors, *Proceedings of the ACM CIKM Conference on Information and Knowledge Management*, pages 776–783, Bremen, Germany, Nov. 2005.

[10] N. Lester, J. Zobel, and H. Williams. In-place versus re-build versus re-merge: Index maintenance strategies for text retrieval systems. In V. Estivill-Castro, editor, *Proceedings of the Australasian Computer Science Conference*, pages 15–22, Dunedin, NZ, Jan. 2004.

[11] J. E. B. Moss. Design of the Mneme persistent object store. *ACM Transactions on Information Systems*, 8(2):103–139, 1990.

[12] T. Strohman. Dynamic collections in Indri. Technical Report IR-426, University of Massachusetts Amherst, 2005.

[13] T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: A language model-based search engine for complex queries. In *IA 2005: Proceedings of the 2nd International Conference on Intelligence Analysis*, 2005.

[14] A. Tomasic, H. García-Molina, and K. Shoens. Incremental updates of inverted lists for text document retrieval. In *SIGMOD 1994: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 289–300, Minneapolis, Minnesota, 1994.

[15] J. Zobel, H. E. Williams, and S. Kimberley. Trends in retrieval system performance. In *ACSC*, pages 241–248, 2000.

PF/Tijah: text search in an XML database system

Djoerd Hiemstra, Henning Rode, Roel van Os and Jan Flokstra
University of Twente
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE Enschede
The Netherlands
{d.hiemstra, h.rode, r.vanos, j.flokstra}@cs.utwente.nl

ABSTRACT

This paper introduces the PF/Tijah system, a text search system that is integrated with an XML/XQuery database management system. We present examples of its use, we explain some of the system internals, and discuss plans for future work. PF/Tijah is part of the open source release of MonetDB/XQuery.

Keywords: Information Retrieval, XQuery, XML Databases, Open Source Systems

1. INTRODUCTION

PF/Tijah is a research project run by the University of Twente with the goal to create a flexible environment for setting up search systems by integrating the PathFinder (PF) XQuery system [3] with the Tijah XML information retrieval system [11]. The PF/Tijah system is part of the open source release of MonetDB/XQuery developed in cooperation with CWI Amsterdam and the University of München. The system is available from SourceForge.[1]

PF/Tijah will include out-of-the-box solutions for common tasks like index creation, stemming, result ranking (supporting several retrieval models), and relevance feedback, but it remains the same time open to any adaptation or extension. On the one hand, the system aims to be a general purpose tool for developing information retrieval (IR) end-user applications using XQuery statements with text search extensions. On the other hand, the system aims to be a playground for the information retrieval scientist and advanced user to easily set up and test new search systems. Advanced users can hook in the system at an intermediate level that provides the database scripting language MIL [4] and several pre-defined operations on terms and XML elements called Score Region Algebra operators [12].

[1] http://sourceforge.net/projects/monetdb/
http://monetdb-xquery.org

Second International Workshop on Open Source Information Retrieval (OSIR), 10 August 2006, Seattle, USA.

The PF/Tijah system has a number unique selling points that distinguish it from most other open source information retrieval systems.

- PF/Tijah supports retrieving arbitrary parts of the textual data, unlike traditional information retrieval systems for which the notion of a *document* needs to be defined up front by the application developer. For instance, if the data consist of scientific journals one can query for complete journals, journal issues, single articles, sections from articles or paragraphs with no need to adapt the index or any other part of the system configuration;
- PF/Tijah supports complex scoring and ranking of the retrieved results by means of so-called NEXI queries. NEXI [15] stands for *Narrowed Extended XPath*: a query language similar to XPath that only supports the descendant and the self axis step, but that is extended with a special `about()` function that takes a sequence of nodes and ranks those by their estimated probability of relevance to the query;
- PF/Tijah supports ad hoc result presentation by means of its query language. For instance, when searching for a special issue of a journal, it is easy to print any information from that retrieval result on the screen in a declarative way (i.e., not by means of a general purpose programming language), such as the special issue title, its date, the editors and the preface. This is simply done by means of XQuery element construction;
- PF/Tijah supports text search combined with traditional database querying, including for instance joins on values. For instance, one could search for employees from the financial department that also worked for the sales department and that sent an email about "tax refunds" (for an example, see below).

This paper is organised as follows. Section 2 presents several usage examples. Section 3 discusses practical design decisions and open problems. Section 4 concludes the paper.

2. SIMPLE USAGE EXAMPLES

PF/Tijah comes with several functions for indexing and retrieving XML data and for managing collections of XML

documents. In this paper, we will focus on the search functionality. We present two usage scenarios: Performing IR-only queries, and performing combined IR and database queries.

2.1 Performing IR-only queries

In order to understand how to use PF/Tijah, it is necessary to have some basic knowledge about using XQuery as for instance described by Katz et al. [9]. XQuery is a database query language for XML data that provides similar functionality as SQL does for relational data. XQuery is built around XPath, a simple path query language. In XPath, a path query such as `/html/head/title` selects from the root of an XML document first all elements tagged as `<html>`, then inside those all tagged as `<head>`, and finally inside those elements all `<title>` elements. PF/Tijah extends XQuery with a small number of user-defined functions, mainly the function `tijah-query()` that takes a sequence of nodes and a NEXI query and produces a sequence of nodes that is ranked by the estimated relevance to the query. NEXI [15] is a path language that supports an additional `about()` function that performs information retrieval queries on XML elements.

At several points in this paper, we will compare PF/Tijah to the draft XQuery full-text search standard that is currently under development by the World Wide Web Consortium [1]. PF/Tijah fulfills many goals of that standard and we might support parts of XQuery full-text in the future. Instead of putting all text search functionality inside user-defined functions as done by PF/Tijah, XQuery full-text extends XQuery with new language constructs for text search. This allows for a more natural way of embedding text search into XQuery. However, the XQuery full-text draft has problems as well: As shown below, XQuery full-text makes the end-user partly responsible for result scoring and ranking.

We will not explain XQuery, XQuery full-text, XPath and NEXI any further in this paper; instead we show some of the functionality of PF/Tijah by example queries, for instance:

```
let $c := doc("mydata.xml")
for $res in tijah-query($c, "//html[about(., ir db)];")
return $res/head/title
```

This query searches for all XML elements tagged as `<html>` using the text query "ir db". The query returns a ranked list of *titles* from those elements.[2]

The following example is slightly more complicated, and involves searching in two different parts of the data. Suppose we are interested in the following: *Give me paragraphs about XQuery in web pages about IR and DB.* This might be expressed as:

```
let $c := doc("mydata.xml")
for $res in tijah-query($c,
  "//html[about(., ir db)]//p[about(., xquery)];")
return $res
```

The query returns a ranked list of paragraphs. The top ranked paragraphs are most likely about "XQuery" and they are contained by web pages that are most likely about "IR" and "DB". So, the ranking is based on a final score that combines scores from the information retrieval query on `<html>` elements with the query on `<p>` elements. PF/Tijah combines scores of multiple `about()` functions in a – for the end user – transparant way, that is, the end user does not have to worry about scoring, score combination, and ranking. Using the W3C XQuery full-text standard, the query above might be expressed as:

```
(:  XQuery full-text instead of PFTijah :)
let $c := doc("mydata.xml")
for $res score $s in $c//p[. ftcontains "xquery" and
  ./ancestor::html ftcontains "ir" && "db"]
order by $s
return $res
```

Here, the user needs to explicitly tell the system to score the query results (by `score $s`), and the user needs to explicitly tell the system to rank the results (by `order by $s`).

By embedding NEXI into XQuery, PF/Tijah follows a different philosophy. In our opinion, result ranking is the *single most important requirement* of a search extension. Therefore, the default behaviour of a PF/Tijah text search query is to return a ranked sequence of nodes. Of course, the default behaviour might still be changed by using the `order by` clause, for instance if the user wants to order the results by their date of publication. In XQuery full-text, the default behaviour seems to be to return all matching nodes in document order, which is probably suboptimal in most applications.

An interesting question is how to combine the scores resulting from both `about()` functions in the PF/Tijah example query above. There are several acceptable ways to do so. Experiments have shown that the best methods for combining and propagating scores depend on the retrieval model that is used to calculate the scores in the first place [13]. PF/Tijah's default behaviour uses so-called language modeling scoring and the best performing combination and propagation methods for this model based on [13]. Advanced users can configure PF/Tijah such they can change the default behaviour of scoring; See Section 3.4. However, there are several problems with PF/Tijah's NEXI extension as well. Section 3.5 will address these further.

Currently, the old Tijah system is used mainly by ourselves, i.e., researchers that participate in scientific search evaluations organised by for instance the Text Retrieval Conference (TREC) [18] and the Initiative for the Evaluation of XML retrieval (INEX) [10]. We are aiming to support the same functionality with PF/Tijah to test for instance the effectiveness of information retrieval models, such as language modeling approaches. As an example, consider the evaluation task organised in the TREC video workshops [17], that provide a collection of videos and MPEG-7 XML annotations on those videos. The MPEG-7 annotations include text annotations from large-vocabulary speech recognition on the audio and optical character recognition on the video. The following query, taken from TRECVid topic 149, returns the identifiers of the top 1000 shots (tagged

[2]For historical reasons, the current implementation requires NEXI queries to end with a semi-colon

as 'videosegment') that contain Condoleeza Rice and their scores.

```
<videoSearchTopicResult tNum="0149"> {
  let $c := doc("trecvid.xml")
  let $q := "//VideoSegment[about(.//TextAnnotation,
    condoleeza rice)];"
  let $result := tijah-query-id($c, $q)
  for $node at $rank in tijah-nodes($result)
  where $rank <= 1000
  return <item seqNum="{$rank}" shotId="{$node/@id}"
    score="{tijah-score($result, $node)}" /> }
</videoSearchTopicResult>
```

In the example above, the returned item elements contain an attribute *score*.[3] Returning the scores of each node in a node sequence is a challenging problem when integrating XQuery with text search functionality. The XQuery data model supports sequences of simple types, but no sequences of complex types such as a sequence of sequences or a sequence of node/score pairs. In order to make scores available to the user, we introduce two functions: First, `tijah-queryid()` which executes the query and returns a result container (actually, the container is just an identifier, hence the name `tijah-queryid()`) that contains a sequence of scored nodes. Second, `tijah-nodes()` which takes the result container and returns a ranked sequence of nodes. So, the statement `tijah-nodes(tijah-query-id())` is equal the function `tijah-query()` introduced previously. The container, however, gives access to the scores of nodes as well by means of the function `tijah-score()`, which gives access to the scores of each retrieved element by returning the score of a node (or by returning a sequence of scores for a node sequence).

The use of separate score and node sequences is advocated by the XQuery full-text standard as well. The following expression of a `for` clause that contains an XQuery full-text `score` variable provides a nice integration of node selection and node scoring:

```
(:  XQuery full-text instead of PFTijah :)
for $result score $score in Expr
...
```

Currently, our system does not support any special language constructs, i.e., our queries are pure XQuery with user-defined functions. We might however support XQuery full-text syntax by evaluating the above as though it is replaced by the following expression, where `$id` and `$score` are new variables not appearing elsewhere:

```
let $id := tijah-query-id(Expr)
for $result in tijah-nodes($id)
let $score := tijah-score($id, $result)
...
```

We believe most users will never actually need these constructs, because the actual scores are often unimportant: It is the *ranking* of results by their score that is important. We like to stress again that PF/Tijah's text search extensions return ranked results; The user does not have to rank the results explicitly using `order by`, so there is often no need to have the actual scores of the retrieved nodes.

The XQuery-fulltext standard [1, Section 4.3.2] suggests another interpretation of their syntax for returning scores. They suggest to evaluate the expression as though it is replaced by:

```
(:  XQuery full-text instead of PFTijah :)
let $scoreSeq := fts:scoreSequence(Expr)
for $result at $rank in Expr
let $score := $scoreSeq[$rank]
...
```

Here as well, `$scoreSeq` and `$rank` are new variables, not appearing elsewhere. In this case, the query without the W3C language extension of the `for` clause results in a rather awkward XQuery statement which includes the text search query "`Expr`" twice.

2.2 Performing combined IR and XML database queries

PF/Tijah makes it possible to process queries that combine the results of an IR part (e.g. a text search query expressed in NEXI) and an DB part (expressed in XQuery). A simple example query for this case might perform a restrictive database selection, for instance give me all documents written by a certain author, and do an IR query only on the publications of that author, so *From a database with research papers, give me titles of documents that the author Vojkan Mihajlovic wrote about "open source XML retrieval".*, which might be expressed as:

```
let $vm := doc("source1.xml")/DOC[author =
  "Vojkan Mihajlovic"]
for $res in tijah-query( $vm,
  "//text[about(.,open source xml retrieval)];")
return $res/title/text()
```

Similar functionality is provided by many other search engines, i.e., this will rank papers about "open source XML retrieval" on top, but the ranking includes only the papers by Vojkan Mihajlovic.

More advanced examples might use joins. A join combines information from two or more items in the database by putting together those items that share the same value for a data item. Query processing in relational databases heavily depends on joining tables on the values of columns. An example of a query that probably cannot be processed by most of todays search engines is the following: *From Shakespeare's plays, give me all speakers from the second part of Henry the Sixth, that speak about a "bloody murder", and that also spoke in first part of Henry the Sixth.* This might be formulated as:

```
for $doc in tijah-query(doc("hen_vi_2.xml"),
  "//SPEECH[about(.,bloody murder)];")
where $doc//SPEAKER = doc("hen_vi_1.xml")//SPEAKER
return $doc//SPEAKER
```

This query ranks the requested speakers by the probabil-

[3]For TRECVid, the score of an item does not need to be returned, but we included it here for illustrative purposes.

ity that they in fact talked about a bloody murder. Such a query might seem a bit far-fetched for searching Shakespeare's plays, but similar queries would be helpful in enterprise search scenarios. For instance, suppose the chief executive officer of a large multinational looks for people that have experience at both the financial department and the sales department and are an expert on "sales tax refunds". He or she might search the enterprise data for the following: *From our enterprise database, give me all employees from the financial department, that sent an email about "sales tax refunds" and that also worked for the sales department.* Obviously, this would be formulated in a similar way as the Shakespeare query above.

As another example, consider the following information need taken from INEX topic 14 [10]: *Find figures that describe the Corba architecture and the paragraphs that refer to those figures. Retrieved components should contain both the figure and the paragraph referring to it.* This might for instance be expressed as:

```
let $doc := doc("inex.xml")
for $p in tijah-query($doc,
  "//p[about(.,corba architecture)];")
for $fig in $p/ancestor::article//fig
where $fig/@id = $p//ref/@rid
return <result> { $fig, $p } </result>
```

Interestingly, the figures and the paragraphs that refer to them might be relatively far apart in the XML data. This query will find figures that do not mention "corba" and "architecture" in their caption (for instance the caption might contain "Object management architectural overview"), but that do mention "corba architecture" in the paragraph when referring to the figure. INEX topic 14 was released in 2002, but was at that time considered to be too difficult by the organisers, or at least, topic 14 at that time needed functionality that none of the participating systems implemented. Another problem with such queries is that they might be hard to assess, that is, it might be hard to determine what elements are good answers. Although there has been an INEX conference each year since 2002, INEX has not taken up the challenge of join queries until today.

3. DESIGN ISSUES AND OPEN PROBLEMS

Started as completely independent academic research projects, Pathfinder and Tijah followed quite different goals – the first one to become an efficient XQuery compiler, the second to work as an XML IR engine. However, both systems share similar internal data models of the XML content and both have been implemented and tested mainly to work on the MonetDB backend. PF/Tijah is a module of Pathfinder that is based on the old Tijah system and developed by we following six general design constraints in order of their importance:

1. Keep Pathfinder untouched by our extension;
2. Use existing Pathfinder functionality as much as possible;
3. Support generalized retrieval methods instead of specialized (document) retrieval;
4. Enable fast retrieval;
5. Minimize redundant storage of data;
6. Use existing Tijah functionality as much as possible.

3.1 Putting things together

PF/Tijah is compiled as a module in Pathfinder. We are using the following elements from Pathfinder:

The document shredder: when loading documents into collections, these documents are shredded by the Pathfinder shredder. This way we inherit all of Pathfinder's document loading properties: speed, loading from any URL, caching, DTD processing, etc.

The XML serializer: when documents have been shredded by Pathfinder, they must be indexed by PF/Tijah. We have constructed a new driver for the existing document serializer in Pathfinder that creates the PF/Tijah index.

The loop-lifted descendant step implementation: instead of using Tijah's containment join implementation we re-use the Pathfinder descendant-step implementation, to minimize the size of the PF/Tijah module. Furthermore, PathFinder staircase joins might be faster than the old Tijah containment joins as they have been well-studied [5, 7].

We are using the following elements from Tijah:

The NEXI query processor: Tijah uses the NEXI query language to express structured information retrieval queries. The module translates NEXI queries into SRA expressions.

Score Region Algebra (SRA): SRA is an algebra for expressing structured information retrieval queries at a logical level [12], i.e, between the query language (conceptual layer) and the database engine and index (physical layer). SRA expressions are independent of the database back-end or storage scheme that is used.

Pathfinder and the old Tijah system use a similar datamodel. Pathfinder uses a pre-/post-order encoding of the XML documents [3], whereas Tijah uses a so-called region data model, i.e. a start-end encoding of the XML document [6], where the region starts are in pre-order and the region endings in post-order. However, PF/Tijah cannot directly run on the Pathfinder index. The main difference is the need to index words instead of nodes. Every occuring word has to be assigned to a new pre-order identifier, not like in Pathfinder where a whole text node is regarded as one unit. PF/Tijah therefore splits text strings into word tokens while keeping their order with the pre-order numbering. This allows to perform simple keyword queries as well as phrase and proximity search.

For integrating the 2 systems, the following points took the most consideration:

PF-light index: The index structure of PF/Tijah is a light version of the Pathfinder index. It is created next to the existing Pathfinder index, which is still used for the normal XQuery evaluation. Apart from indexing single words here instead of whole text nodes, it knows additional inverted structures sorted on term identifiers

to accelerate the selection of term occurrences in the collection. On the other hand, it is still a *light* index in the sense that it doesn't replicate the full document content. Attributes, Processing Instructions, etc. are kept in the normal Pathfinder index only, since they will not be addressed with NEXI queries.

A further problem arising, working with 2 different indices, is the translation between both. A sequence of nodes created by an arbitrary XQuery expression that is further used in a search process, has to be translated from Pathfinder pre-order identifiers to their corresponding ones in the PF-light index. The same vice versa for passing the ranked result sequence back and assigning them to an XQuery variable.

SRA operator implementations: All SRA operators needed a new implementation on the physical layer now working on the new PF-light index. Special attention was given here to enable fast term selection and to employ the highly performance trimmed containment-join operators coming with the Pathfinder system.

3.2 The MonetDB backend

In contrast to many other open source IR environments like Lucene [8], Lemur/Indri [14], or Terrier [16] our system is set up on a light database backend, the MonetDB system. Whereas pure IR systems can profit from highly specialized algorithms, PF/Tijah gains a lot flexibility by abstraction from the physical layer. Search extensions are easily programmed as database scripts, pre- and post-processing of the data can be done without the need to extend the system. Furthermore, since database operations are highly tuned to efficient data processing, many IR operations can keep up with, or even beat, the efficiency of specialized IR systems.

The used database backend, MonetDB [4], is an open source main memory database system. It is a light-weight system compared to common commercial products, but allows highly effective data processing. If required, new database operations can be added to the database kernel. The main limitation of the system is its restriction to process queries completely in main memory. Our IR tools manage collections up to several gigabytes on a single machine, currently. This limitation will be overcome by a new version of the database backend [19].

3.3 Collection handling

PF/Tijah comes with additional functionality for handling collections of a large number of smaller XML documents. This functionality includes naming a collection (or database) adding a document to the collection, and *finalizing* the collection, that is, building the inverted files. Additionally, PF/Tijah can be configured in such a way that the XML data is not loaded in the Pathfinder system at index time. In this way, only PF/Tijah's Pathfinder light index is built. At query time, if for instance information has to be returned of a top 10 retrieved by a NEXI query, Pathfinder loads only those XML documents that are actually returned by the query.

3.4 System configuration

The functions `tijah-query()` and `tijah-query-id()` allow the inclusion of any number of options by means of a special empty XML element `<TijahOptions/>` as follows.

```
let $opt := <TijahOptions algebraType="COARSE2"
  txtmodel_model="NLLR" />
let $c := doc("mydata.xml")
for $res in tijah-query($c, "//html[about(., scoring)];")
return $res//title
```

With these options, the user can overwrite the default behaviour for query plan generation (using the `algebraType` attribute), for scoring in an about clause (using the attribute `txtmodel_model`), but also for score combination and propagation, etc.

3.5 Problematic effects of the NEXI language embedding

Currently, NEXI queries are embedded as strings in XQuery expressions. The NEXI query string remains a black box for XQuery, so no variable usage or static type checking and syntax checking is possible in XQuery. A tighter integration is desirable, for instance as recommended by the XQuery full-text standard.

Another problem is that NEXI and XQuery share some expressiveness. Both formalisms are able to process simple path queries. For example, the following queries are currently equivalent:

```
let $c := doc("test.xml")
return tijah-query($c//DOC//TEXT, "//P[about(.,XML)];")
```

and

```
let $c := doc("test.xml")
return tijah-query($c, "//DOC//TEXT//P[about(.,XML)];")
```

In future versions of the system, the text search functions could be less expressive. This however, might introduce problems with scoring complex queries as shown in Section 2.1.

4. CONCLUSIONS

PF/Tijah aims at supporting search applications for highly structured, heterogenous content on medium-sized XML collections. The system can be used to quickly develop retrieval applications. It is part of the open source release of MonetDB/XQuery (version 0.13.1 and higher) and available from SourceForge.

To develop PF/Tijah, we made choices that are currently not recommended by the XQuery full-text draft standard. Our choices are driven by a focus on *ranking* search results. In our opinion, result ranking is the *single most important* requirement for text search. If the ranking procedures are not effective, advanced features like implicit sentences and paragraphs, thesaurus queries, queries across elements, etc. will be of limited use. Currently, XQuery full-text only gives limited thought to scoring. In the W3C use cases draft [2] only one of 16 use case sections is devoted to result ranking. The other sections assume that elements of text search queries are returned in document order.

Note that we do not advocate to define scoring in any way by the standard: On the contrary, scoring must be implementation-defined, and users should not be bothered by explicit

ranking of the results using XQuery `order by` and not be bothered by explicit combination of scores: The `tijah-query()` function of the PF/Tijah module returns a *ranked* sequence of nodes, and the NEXI query language supports simple axis steps to enable powerful score combination.

For research at the University of Twente, PF/Tijah serves as our "Swiss knife" for research into information retrieval and research into the integration of information retrieval and databases. With PF/Tijah, it is relatively easy to experiment with new retrieval approaches, and it is relatively easy to test the implications and limitations of approaches to XML information retrieval: those advocated by XQuery full-text as well as those advocated by for instance the INEX workshops.

More information on PF/Tijah is currently available from the project wiki at: http://monetdb.cwi.nl/projects/trecvid/MN5/index.php/PFTijah_Wiki

Acknowledgements

This research is funded by the Dutch BSIK progamme MultimediaN. Many thanks to Vojkan Mihajlović, Thijs Westerveld (CWI Amsterdam), Georgina Ramírez (CWI Amsterdam), and Arjen de Vries (CWI Amsterdam) for helpful advice and for testing the system.

5. REFERENCES

[1] S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, M. Holstege, D. McBeath, M. Rys, and J. Shanmugasundaram. XQuery 1.0 and XPath 2.0 full-text. Technical report, Word Wide Web Consortium. working draft 1 May 2006, http://www.w3.org/TR/xquery-full-text

[2] S. Amer-Yahia and P. Case. XQuery 1.0 and XPath 2.0 full-text use cases. Technical report, Word Wide Web Consortium. working draft 1 May 2006, http://www.w3.org/TR/xmlquery-full-text-use-cases/

[3] P.A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and Jens Teubner. MonetDB/XQuery: A fast XQuery processor powered by a relational engine. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2006.

[4] P.A. Boncz. *Monet: A Next-Generation DBMS Kernel for Query-Intensive Applications*. PhD thesis, University of Amsterdam, 2002.

[5] P.A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. Loop-lifted staircase join: from XPath to XQuery. Technical Report INS-E0510. *CWI, Amsterdam*, 2005.

[6] F.J. Burkowski. Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Texts. In *Proceedings of the 15th ACM SIGIR Conference on Research and Development in Information Retrieval*, 1992.

[7] T. Grust, M. van Keulen, and J. Teubner. Staircase join: Teach a relational DBMS to watch its (axis) steps. In *Proceedings of the 29th Conference on Very Large Databases (VLDB)*, 2003.

[8] E. Hatcher and O. Gospodnetic. *Lucene in action*. Manning Publications, 2005.

[9] H. Katz, D. Chamberlin, D. Draper, M. Fernandez, M. Kay, J. Robie, M. Rys, J. Simeon, J. Tivy, and P. Wadler. *XQuery from the Experts: A Guide to the W3C XML Query Language*. Addison Wesley, 2003.

[10] M. Lalmas and G. Kazai. Report on the ad-hoc track of the INEX 2005 workshop. *SIGIR Forum* 40(1):49–57, 2005.

[11] J. List, V. Mihajlovic, G. Ramirez, A.P. de Vries, D. Hiemstra, and H.E. Blok. Tijah: Embracing information retrieval methods in XML databases. *Information Retrieval Journal* 8(4):547–570, 2005.

[12] V. Mihajlovic. Score region algebra: A framework for structured information retrieval. In *SIGIR Doctoral Consortium Workshop*, 2005.

[13] V. Mihajlovic, H.E. Blok, D. Hiemstra, and P.M.G. Apers, Score Region Algebra: Building a Transparent XML-IR Database. In *Proceedings of the 14th International Conference on Information and Knowledge Management (CIKM)*, 2005

[14] P. Ogilvie and J. Callan. Experiments using the Lemur toolkit. In *Proceedings of the tenth Text Retrieval Conference (TREC)*, 2001.

[15] R. A. O'Keefe and A. Trotman. The simplest query language that could possibly work. In *Proceedings of the 2nd Initiative for the Evaluation of XML Retrieval (INEX)*. ERCIM workshop proceedings, 2004. http://www.cs.otago.ac.nz/postgrads/andrew/2003-4.pdf.

[16] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and D. Johnson. Terrier information retrieval platform. In *Proceedings of the 27th European Conference on Information Retrieval (ECIR)*, 2005.

[17] A.F. Smeaton, P. Over, and W. Kraaij. TRECvid: evaluating the effectiveness of information retrieval tasks on digital video. In *Proceedings of ACM Multimedia*, pages 652–655, 2004.

[18] E.M. Voorhees and D. Harman, editors. *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, 2005.

[19] M. Zukowski, S. Heman, A.P. de Vries, and P. Boncz. Efficient and Flexible Information Retrieval Using a Relational Database Engine. *submitted*, 2006.

Terrier: A High Performance and Scalable Information Retrieval Platform

Iadh Ounis, Gianni Amati*, Vassilis Plachouras, Ben He,
Craig Macdonald, Christina Lioma
Department of Computing Science
University of Glasgow Scotland, UK
{ounis,gianni,vassilis,ben,craigm,xristina}@dcs.gla.ac.uk

ABSTRACT

In this paper, we describe Terrier, a high performance and scalable search engine that allows the rapid development of large-scale retrieval applications. We focus on the open-source version of the software, which provides a comprehensive, flexible, robust, and transparent test-bed platform for research and experimentation in Information Retrieval (IR).

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Design, Experimentation, Performance, Theory

Keywords

Terrier, Information Retrieval platform, Open Source

1. INTRODUCTION

Terrier [12], Terabyte Retriever, is a project that was initiated at the University of Glasgow in 2000, with the aim to provide a flexible platform for the rapid development of large-scale Information Retrieval applications, as well as a state-of-the-art test-bed for research and experimentation in the wider field of IR.

The Terrier project explored novel, efficient and effective search methods for large-scale document collections, combining new and cutting-edge ideas from probabilistic theory, statistical analysis, and data compression techniques. This led to the development of various retrieval approaches using a new and highly effective probabilistic framework for IR, with the practical aim of combining efficiently and effectively various sources of evidence to enhance the retrieval performance.

In particular, we have developed several new hyperlink structure analysis methods [15], various selective combination of evidence approaches [13, 16], new document length normalisation methods [1, 7, 8], many automatic query expansion and re-formulation techniques [1, 11], as well as a comprehensive set of query performance predictors [6, 9], to support a wide range of retrieval tasks and applications.

In addition, an effective in-house, highly configurable and scalable crawler called Labrador[1] has been developed. The crawler was used for the deployment of Terrier in various industrial applications. Various powerful compression techniques have been explored and implemented, making Terrier particularly suitable for large-scale collections. Finally, the project investigated distributed architectures and retrieval [4], which allowed the system to be used both in a centralised and distributed setting.

This paper focuses on a core version of Terrier, an open source product, which has been available to the general public since November 2004 under the MPL license[2]. The open source version of Terrier is written in Java, allowing the system to run on various operating systems platforms, and a wide variety of hardware. Terrier, the latest version of which is 1.0.2, is available to download from `http://ir.dcs.gla.ac.uk/terrier/`.

2. MOTIVATIONS & AIMS

In any experimental science field, as is the case in IR, it is crucial to have a system allowing large-scale experimentation to be conducted in a flexible, robust, transparent and reproducible way. Terrier addresses this important issue by providing a test-bed framework for driving research and facilitating experimentation in IR.

The major leading IR groups in the world have always had an experimental platform to support their research. Most of the existing IR platforms are designed towards supporting a particular IR method/approach, making them difficult to customise and tailor to new methods and approaches.

Terrier is the first serious answer in Europe to the dominance of the United States on research and technological solutions in IR. Our objective was to include the state-of-the-art IR methods and techniques, and offer the most efficient and effective IR technologies, into a transparent, easily extensible and modular open source software. The open source product of Terrier was therefore designed as a tool to evaluate, test and compare models and ideas, and to build systems for large-scale IR.

The system includes efficient and effective state-of-the-art retrieval models based on a new Divergence From Randomness (DFR) framework for deriving parameter-free proba-

*Also affiliated to Fondazione Ugo Bordoni, Rome

SIGIR Open Source Workshop '06 Seattle, Washington, USA.

[1] `http://ir.dcs.gla.ac.uk/labrador/`
[2] `http://www.mozilla.org/MPL/`

bilistic models for IR [1]. The DFR models are information-theoretic models, for both query expansion and document ranking, a versatile feature that is not possessed by any other IR model. In addition, for easy cross-comparison of different retrieval models, Terrier includes a wide variety of models, ranging from numerous forms of the classical TF-IDF weighting scheme to the recent language modelling approach, through the well-established Okapi's BM25 probabilistic ranking formula.

Terrier provides various indexing and querying APIs, and allows rapidly experimenting with new concepts/ideas on various collections, and in different configuration settings. This important feature for a research platform is made possible by the modular architecture of the system, its various easy-to-use configuration options, as well as its use of the most recent software engineering methods. The system is very easy to work with, further helped with a comprehensive documentation[3], and is easy to extend and adapt to new applications. This is demonstrated by the use of the platform in such diverse search tasks as desktop search, web search, e-mail & expertise search, XML retrieval, multilingual and blogs search.

Finally, Terrier includes various retrieval performance evaluation tools, which produce an extensive analysis of the retrieval effectiveness of the tested models/concepts on given test collections. Systems and their variants can be instantiated in parallel, an important feature for conducting numerous experiments at the same time, thus fostering extensive experimentation in a structured and systematic way.

The remainder of this paper is structured as follows. We describe the main indexing features of Terrier in Section 3, and their associated data structures in Section 4. Section 5 provides an overview of the retrieval features of Terrier, which make the system particularly suitable for research purposes. In Section 6, we describe a very interesting functionality of Terrier, namely query expansion, and outline how it can be used for other IR activities and applications. The retrieval performance evaluation package of Terrier is presented in Section 7, while some "out-of-the-box" applications of Terrier are described in Section 8. Finally, we conclude the paper with lessons learnt while building the Terrier platform.

3. INDEXING FEATURES

Figure 1 outlines the indexing process in Terrier. Terrier is designed to allow many different ways of indexing a corpus of documents. Indexing is a four stage process, and at each stage, plugins can be added to alter the indexing process. This modular architecture allows flexibility in the indexing process at several stages: in the handling of a corpus of documents, in handling and parsing each individual document, in the processing of terms from documents, and in writing the index data structures. In addition, Terrier allows the direct parsing and indexing of compressed collections.

The open source version of Terrier comes with support for indexing test collections from the well-established TREC[4] international evaluation forum. This permits researchers to quickly set up and run Terrier with a test collection, lessening the learning curve for the platform. The advantage is that any other storage method for a collection of documents

[3] `http://ir.dcs.gla.ac.uk/terrier/docs.html`
[4] `http://trec.nist.gov`

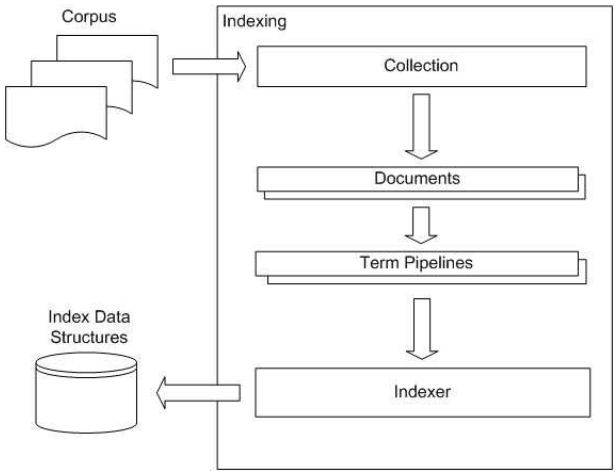


Figure 1: Overview of the indexing architecture of Terrier. A corpus of documents is handled by a Collection plugin, which generates a stream of Document objects. Each Document generates a stream of terms, which are transformed by a series of Term Pipeline components, after which the Indexer writes to disk.

can easily be supported by adding a new Collection plugin. For example, reading a stream of documents from an email server would only require implementing a new Collection plugin to connect to the email server.

Terrier comes with various document parsers embedded. These include the ability to index HTML documents, Plain text documents, Microsoft Word, Excel and Powerpoint documents, as well as Adobe PDF files. To add support for another file format to Terrier, a developer must only add another Document plugin which is able to extract the terms from the document.

Each term extracted from a document has three fundamental properties: firstly, the actual String textual form of the term; secondly, the position at which the term occurs in the document, and thirdly, the fields in which the term occurs (fields can be arbitrarily defined by the Document plugin, but typically are used to define which HTML tags a term occurs in). Terrier adds configurability at this stage of the indexing: terms are passed through a 'Term Pipeline', which allows the terms to be transformed in various ways. Terrier comes with some pre-defined Term Pipeline plugins, such as two variants of Porter's stemming algorithm, and a stopwords removal component. Term Pipeline plugins introduce flexibility to the processing and transformation of terms, which can be manifested for example by n-gram indexing, adding stemming and stopword removal in various languages, acronym expansion, or use of a thesaurus, to name but a few.

The last component in the Term Pipeline chain is always the Indexer. The Indexer is responsible for writing the index using the appropriate data structures. By using a Block Indexer, it is possible to create an index that stores the positions of terms in each document. Each document is divided into blocks - the size of a block defines the accuracy with which term positions are recorded. By default, the size of a block is 1 and, in that case, the exact positions of the

term occurrences are recorded. This allows for proximity and phrase operators to be used in queries during retrieval. However, a block could also be defined as a semantic entity, so as to allow structured retrieval. The following section describes the index data structures generated by Terrier when indexing a corpus of documents.

4. INDEX STRUCTURES

A Terrier index consists of 4 main data structures, in addition to some auxiliary files:

- **Lexicon:**
  The lexicon stores the term and its term id (a unique number for each term), along with the global statistics of the term (global term frequency and document frequency of the term) and the offsets of the postings list in the Inverted Index.

- **Inverted Index:**
  The inverted index stores the postings lists of a term. In particular, for each term, the inverted index stores: the document id of the matching document; and the term frequency of the term in that document. The fields in which the term occurred are encoded using a bit set. If the index has been made with positional information, the postings list will also contain the block ids in the document that the term occurs in. Positional information allows phrasal and proximity search to be performed.

  The postings list in Terrier is highly compressed. In particular, the document ids are encoded in the inverted index using Gamma encoding, the term frequencies are encoded using Unary encoding, and the Block ids, if present, are encoded using Gamma encoding [20].

- **Document Index:**
  The Document Index stores the document number (an external unique identifier of the document), the document id (internal unique identifier of the document); the length of the document in terms of tokens; and the offset of the document in the Direct Index.

- **Direct Index:**
  The Direct Index stores the terms and term frequencies of the terms present in each document. The aim of the Direct Index is to facilitate easy and efficient query expansion, as described in Section 6. However, the direct index is also extremely useful for applying clustering to a collection of documents.

  The direct index contents are compressed in an orthogonal way to the inverted index. Term ids are written using Gamma encoding, term frequencies use Unary encoding and the fields in which the terms occur are encoded using a bit set. Block ids, if present, are encoded using Gamma encoding.

Table 1 details the format of each file in the index structures. Moreover, Table 2 shows the sizes of Terrier's index data structures after indexing the WT2G collection. Fields are not recorded, while index sizes with and without term positions are stated. For comparison purposes, the index sizes of Terrier without compression, and two other open source products, Indri[5] and Zettair[6] for the same collection are also provided. Like Terrier, Indri and Zettair are deployed using their default settings.

Index Structure	Contents
Lexicon	Term (20) Term id (4) Document Frequency (4) Term Frequency (4) Byte offset in inverted file (8) Bit offset in inverted file (1)
Inverted Index	Document id gap (gamma code) Term Frequency (unary code) Fields (# of fields bits) Block Frequency (unary code) [Block id gap (gamma code)]
Document Index	Document id (4) Document Length (4) Document Number (20) Byte offset in direct file (8) Bit offset in direct file (1)
Direct Index	Term id gap (gamma code) Term frequency (unary code) Fields (# of fields bits) Block frequency (unary code) [Block id gap (gamma code)]
Lexicon Index	Offset in lexicon (8)
Collection Statistics	# of documents # of tokens # of unique terms # of pointers

Table 1: Details on the format and compression used for each index data structure. The numbers in parenthesis are the size of each entry in bytes, unless otherwise denoted. Gamma and unary codes are variable length encodings.

5. RETRIEVAL FEATURES

One of the main aims of Terrier is to facilitate research in the field of IR. Figure 2 provides an outline of the retrieval process in Terrier. Terrier's retrieval features have been selected in order to be useful for a wide range of IR research. Indeed, Terrier offers great flexibility in choosing a weighting model (Section 5.1), as well as in altering the score of the retrieved documents (Section 5.2). Moreover, Terrier offers an advanced query language (Section 5.3). Another very important retrieval feature of Terrier is the automatic query expansion, which is described in Section 6.

5.1 Weighting Models

The core functionality of matching documents to queries and ranking documents takes place in the Matching module. Matching employs a weighting model to assign a score to each of the query terms in a document. In order to facilitate the cross-comparison of weighting models, a range of weighting models is supplied, including BM25, TF-IDF, and document weighting models from the Divergence From Ran-

[5] `http://www.lemurproject.org/indri/`
[6] `http://www.seg.rmit.edu.au/zettair/`

Index Structure	Index Sizes	% Of WT2G
Terrier 1.0.2		
Direct Index	87MB	4%
Inverted Index	72MB	3%
Lexicon	40MB	2%
Document Index	8.8MB	0.4%
Terrier if no compression is used		
Direct Index	481MB	22%
Inverted Index	481MB	22%
Lexicon	40MB	2%
Document Index	8.8MB	0.4%
Terrier 1.0.2 with Term Positions		
Direct Index	366MB	17%
Inverted Index	349MB	17%
Lexicon	40MB	2%
Document Index	8.8MB	0.4%
Indri 2.2		
Direct Index	386MB	18%
Inverted Index	417MB	19%
Zettair 0.6.1		
Inverted Index	430MB	20%

Table 2: Index sizes for Terrier (with and without term positions), Indri and Zettair for the WT2G collection (2.1GB). Terrier's uncompressed index size is provided for comparison purposes.

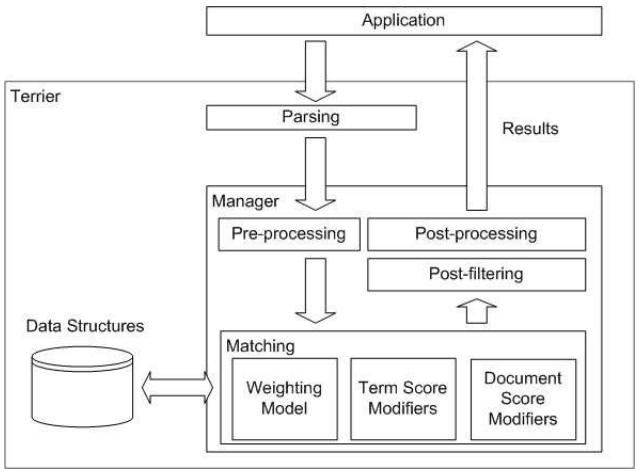


Figure 2: Overview of the retrieval architecture of Terrier. The application communicates with the Manager, which in turn runs the desired Matching module. Matching assigns scores to the documents using the combination of weighting model and score modifiers.

domness (DFR) framework [1]. The DFR approach supplies parameter-free probabilistic models, based on a simple idea:

> *"The more the divergence of the within-document term-frequency from its frequency within the collection, the more the information carried by the term t in the document d".*

The open source 1.0.2 version of Terrier includes eight document weighting models from the DFR framework, all of which perform robustly on standard test collections. Ponte-Croft's language model [18] is also supported.

5.2 Altering Document Scores

The score of an individual term in a document can be altered by applying a TermScoreModifier. For example, a TermInFieldModifier can be applied in order to ensure that the query terms occur in a particular field of a document. If a query term does not appear in a particular field, then the TermInFieldModifier resets the score of the document. In addition, a FieldScoreModifier boosts the score of a document in which a query term appears in a particular field.

Similarly, changing the score of a retrieved document is achieved by applying a DocumentScoreModifier. One such modifier is the PhraseScoreModifier, which employs the position information saved in the index of Terrier, and resets the score of the retrieved documents in which the query terms do not appear as a phrase, or within a given number of blocks. Generally, a DocumentScoreModifier is ideal for applying query-independent evidence, such as evidence from hyperlink structure analysis, or from the URL of documents. Moreover, the selective application of different retrieval techniques based on evidence from the hyperlink structure [13] can be applied as a DocumentScoreModifier.

After the application of any TermScoreModifiers or DocumentScoreModifiers, the set of retrieved documents can be further altered by applying post processing or post filtering. Post processing is appropriate to implement functionalities that require changing the original query. An example of post processing is Query Expansion (QE), which is described in detail in Section 6. The application of QE could be enabled on a per-query basis, before retrieval, depending on the output of a pre-retrieval performance predictor [6]. Another possible example of post processing could be the application of clustering.

Post filtering is the final step in Terrier's retrieval process, where a series of filters can remove already retrieved documents, which do not satisfy a given condition. For example, in the context of a Web search engine, a post filter could be used to reduce the number of retrieved documents from the same Web site, in order to increase diversity in the results.

5.3 Query Language

Terrier includes a powerful query language that allows the user to specify additional operations on top of the normal probabilistic queries. Such operations may specify that a particular query term should or should not appear in the retrieved documents. Other available operations include requiring that terms appear in particular fields, phrase queries, or proximity queries. An overview of the available query language operations is given below.

- `t1 t2` retrieves documents with either `t1` or `t2`.
- `t1^2.3` sets the weight of query term `t1` to 2.3.

- `+t1 -t2` retrieves documents with `t1` but not t2.
- ``t1 t2'' retrieves documents where the terms `t1` and `t2` occur next to each other.
- ``t1 t2''∼n retrieves documents where the terms `t1`, `t2` occur within n blocks.
- `+(t1 t2)` specifies that both terms `t1` and `t2` are required.
- `field:t1` retrieves docs where `t1` must appear in the specified field.
- `control:on/off` enables or disables a given control. For example, query expansion is enabled with `qe:on`.

The query language operations correspond to TermScoreModifier or DocumentScoreModifier modules, which are appropriately configured when the query is parsed. Retrieval parameters can also be specified by the user, when permitted, to change or enable retrieval functionalities. For example, a query for which a particular query term should appear in the title of the retrieved documents, automatically enables a TermInFieldModifier.

6. QUERY EXPANSION

Terrier includes automatic pseudo-relevance feedback, in the form of Query Expansion (QE). The method works by taking the top most informative terms from the top-ranked documents of the query, and adding these new related terms into the query. This operation is made possible by the presence of the direct index. The direct index allows the terms and their frequencies to be determined for each document in the index. The new query is reweighted and rerun - providing a richer set of retrieved documents. Terrier provides several term weighting models from the DFR framework which are useful for identifying informative terms from top-ranked documents. In addition, for easy cross-comparison, Terrier also includes some well-established pseudo-relevance feedback techniques, such as Rocchio's method [19].

Automatic query expansion is highly effective for many IR tasks. The decision of applying QE depends on the type of application, and the difficulty of queries. Therefore, we have developed several tools for determining possible indicators that predict the query difficulty, and the selective application of query expansion [2, 6, 9]. The vocabulary and size of the collections may be largely different. As a consequence, automatic means to tune the inherent parameters of the query expansion are also necessary in a good IR system. For this purpose, we devised a parameter-free QE mechanism, called Bo1 [1, 11], as a default in Terrier. The system is also readily available to support several low-cost methods for a selective application of query expansion. Our method of QE guarantees the most effective and efficient mechanism for performing QE, even with only using 3 retrieved documents and a handful of additional query terms. In addition, QE can be also used to automatically generate realistic samples of queries [6].

Data structures, in particular the direct file, tools and methods adopted for performing QE in Terrier, also constitute an advanced basis for other types of IR activities and applications, such as document and term clustering or question answering. Indeed, the query expansion methodology is based on a filter, activated on the information theoretic weights of the terms, which selects the highly informative terms that appear in at least X retrieved documents. As an illustration, the optimal value for X is 2 in adhoc retrieval. The clustering of the results would require a higher value for X, while question answering may require a different information theoretic definition for X. In conclusion, QE is a good source of inspiration for new and interesting IR applications and research directions.

Collection	Topics/Task	Performance
WT2G	Adhoc Topics 401-450	0.2663 MAP
WT10G	Adhoc Topics 451-550	0.1886 MAP
W3C	Known Item Topics 26-150	0.4960 MRR

Table 3: Performance of "out-of-the-box" Terrier on standard TREC collections and tasks. Performance is measured using the standard measure for each task: for Adhoc, Mean Average Precision (MAP); for Known Item, Mean Reciprocal Rank (MRR).

7. EVALUATION PACKAGE

When performing IR research using a standard test collection, it is essential to evaluate the retrieval performance of the applied approaches. The open source version of Terrier was designed to allow users to rapidly design and test new retrieval functions and IR models. For this reason, we implemented the evaluation tools in the TREC style with a wide range of known IR measures. In particular, Terrier includes and integrates the main evaluation functionalities found in the trec_eval tool, including, among others, calculating Mean Average Precision, Precision @ rank N, interpolated Precision, and R-Precision. It is also possible to show the evaluation measures on a per-query basis.

The evaluation tools of Terrier allow easy testing of new retrieval methodologies or approaches. Once the new approach has been integrated into Terrier, users can test it, together with as many models as possible, against a test collection. As soon as the system has terminated all the runs, for each model, the file of the evaluation results can be created and a summary is displayed.

8. STANDARD DEPLOYMENTS & SAMPLE APPLICATIONS

Terrier 1.0.2 comes with two applications. Firstly, Terrier comprises a complete application suite for performing research using test collections. Terrier is easy to deploy on the standard TREC test collections, e.g. the TREC CDs, WT2G, WT10G, .GOV etc. This facilitates research on these standard test collections. Table 3 shows "out-of-the-box" retrieval performance scores of Terrier on standard test collections, using title-only queries. The default settings include the removal of English stopwords and the application of Porter's stemming algorithm. During retrieval, the InL2 DFR weighting model is applied as is.

Secondly, Terrier comes with a proof-of-concept Desktop Search application, shown in Figure 3. This shows how the retrieval functionalities of Terrier can be easily deployed to an application suitable for day-to-day retrieval tasks. All the retrieval functionalities are easily accessible from the Desktop Search application, including Terrier's query language and query expansion.

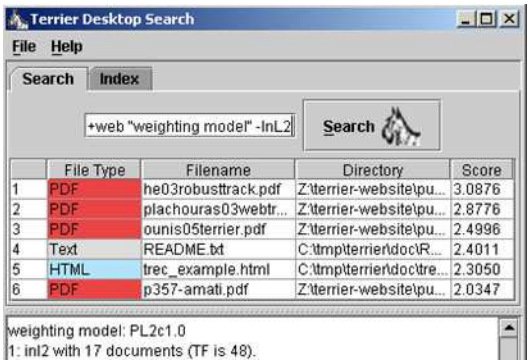


Figure 3: Screenshot of the proof-of-concept Terrier Desktop Search. It can index various types of common document formats, including Microsoft Office documents, HTML and PDF files.

9. RESEARCH FACILITATED & LESSONS LEARNED

The development of a test-bed framework allowing for the rapid experimentation of new IR concepts/ideas has boosted our research, yielding some significant insights into the behaviour of IR techniques on diverse, multilingual, and large-scale collections. The system allowed us to successfully and significantly participate in various TREC tracks such as the Web tracks (2001-2004), the Robust tracks (2003, 2004), the Terabyte tracks (2004-2005), and recently the Enterprise track (2005). In addition, we participated in various tracks of CLEF[7] in 2003-2005.

More importantly, in addition to being a vehicle for the evaluation of our new search methods, the Terrier platform allowed us to easily simulate, assess, and improve state-of-the-art IR methods and techniques. This led to a better understanding of the underlying issues related to the tackling of different search tasks on various and diverse large collections. In particular, we realised the need to build parameter-free models, which avoid the constraint of using costly parameter-tuning approaches based on relevance assessment. The proposed DFR framework is not only an efficient and effective IR model, but also a mechanism by which we can explain and complement other existing IR approaches/techniques [1].

One of the main lessons learnt from building the Terrier platform during the previous years is the fact that building a sustainable, flexible and robust platform for information retrieval is equally important as creating new IR models. Indeed, research in an experimental field such as IR cannot be conducted without a platform facilitating experimentation and evaluation.

In developing a cutting-edge technology from a laboratory setting to a product release, which was deployed in various industrial applications and settings, we gained a much better understanding of the challenges faced in tackling real large-scale collections, not necessarily as clean as the TREC or CLEF collections, and how to provide practical and effective solutions under a high query workload.

Many benefits were achieved by releasing a core version of the project as an open source software. By facilitating a transparent and reproducible experimentation of various retrieval methods, we believe that we achieved a greater impact and visibility in the scientific community. We have also built a community of users/developers around Terrier, which helped improving the quality of the software and attracted more researchers to the project. Since its release in November 2004, the Terrier software has been downloaded thousands of times from across the world, including major commercial search engine companies.

Finally, the Terrier project has attracted many undergraduate, postgraduate, and research students, and benefited from their contributions. Building an IR platform is both a research and engineering process, requiring team work and a critical mass of developers. Terrier is currently the main development platform for both undergraduate and postgraduate students in our research group, allowing them to employ a state-of-the-art framework in their learning and research.

10. CONCLUSIONS

In this paper, we described the open source version of Terrier, a robust, transparent, and modular framework for research and experimentation in Information Retrieval. The platform was developed as part of an ongoing larger project, ultimately aiming at becoming the European Search Engine, with the state-of-the-art search technology for various search tasks and applications.

Terrier is a mature and state-of-the-art platform, including highly effective retrieval models, based on the DFR framework. The performance of these models is at least comparable to, if not better than, that of the most recent models, including on very large-scale test collections [14, 11]. In addition, Terrier's DFR models are information-theoretic models for both query expansion and document ranking, a versatile and original feature that is not possessed by any other IR model. Terrier is continuously expanded with new models resulting from our research. Indeed, new research, such as models for document structure and combination of evidence [17], length normalisation methods [7], query performance prediction [6, 9], NLP techniques [10], information extraction [3], is being readily fed to Terrier. Furthermore, Terrier is being used for research in enterprise and expert search [11]. Retrieval from large-scale test collections has also led us to study optimal distributed architectures for retrieval systems [4, 5].

From an IR perspective, technological challenges may render an IR model obsolete very quickly (as has happened with the Boolean or Vector Space model). We believe that a theoretical framework for Information Retrieval, when implemented within a mature, robust and extensible IR system, is destined to last. Our strong belief is that Terrier, as the accumulator of innovative ideas in IR, and with its design features and underlying principles, open source being one of them, will make easier technology transfers.

11. ACKNOWLEDGEMENTS

The development of Terrier required a significant manpower and infrastructure. We would like to thank those many students and programmers who participated in developing Terrier. We would also like to thank the UK Engineering and Physical Sciences Research Council (EPSRC), project grant number GR/R90543/01, as well as the Leverhulme Trust, project grant number F/00179/S, who have

[7] http://www.clef-campaign.org

partially supported the development of Terrier. Finally, we thank the Department of Computing Science at the University of Glasgow, and Keith van Rijsbergen, for supporting the Terrier platform at various stages of its development.

12. REFERENCES

[1] G. Amati. *Probabilistic Models for Information Retrieval based on Divergence from Randomness*. PhD thesis, Department of Computing Science, University of Glasgow, 2003.

[2] G. Amati, C. Carpineto, and G. Romano. Query Difficulty, Robustness, and Selective Application of Query Expansion. In *ECIR '04: Proceedings of the 26th European Conference on Information Retrieval (ECIR'04)*, pages 127-137, Sunderland, UK, 2004. Springer.

[3] G. Amati. Information Theoretic Approach to Information Extraction. In Proceedings of the 7th international conference on Flexible Query Answering Systems (FQAS 2006), pages 519–529, Milan, Italy, 2006.

[4] F. Cacheda, V. Plachouras, and I. Ounis. A case study of distributed information retrieval architectures to index one Terabyte of text. *Information Processing & Management*, 41(5):1141–1161, 2005.

[5] F. Cacheda, V. Carneiro, V. Plachouras, and I. Ounis. Performance Analysis of Distributed Information Retrieval Architectures Using an Improved Network Simulation Model. *Information Processing & Management* (to appear).

[6] B. He, and I. Ounis. Inferring Query Performance Using Pre-retrieval Predictors. In *Proceedings of the 11th Symposium on String Processing and Information Retrieval (SPIRE 2004)*, Padova, Italy, 2004.

[7] B. He, and I. Ounis. A study of the Dirichlet priors for term frequency normalisation. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and Development in information retrieval*, pages 465–471, Salvador, Brazil, 2004.

[8] B. He, and I. Ounis. Term Frequency Normalisation Tuning for BM25 and DFR Models. In *ECIR '05: Proceedings of the 27th European Conference on Information Retrieval (ECIR'05)*, pages 200–214, Santiago de Compostela, Spain, 2005. Springer.

[9] B. He, and I. Ounis. Query Performance Prediction. In *Information Systems*, Elsevier, 2006 (In press).

[10] C. Lioma, and I. Ounis. Examining the Content Load of Part-of-Speech Blocks for Information Retrieval. In *Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics, (COLING/ACL 2006)*, Sydney, Australia, 2006.

[11] C. Macdonald, B. He, V. Plachouras, and I. Ounis. University of Glasgow at TREC2005: Experiments in Terabyte and Enterprise Tracks with Terrier. In *TREC '05: Proceedings of the 14th Text REtrieval Conference (TREC 2005)*, November, 2005, NIST.

[12] I. Ounis, G. Amati, Plachouras V., B. He, C. Macdonald, and D. Johnson. Terrier Information Retrieval Platform. In *Proceedings of the 27th European Conference on IR Research (ECIR 2005)*, volume 3408 of *Lecture Notes in Computer Science*, pages 517–519. Springer, 2005.

[13] V. Plachouras, and I. Ounis. Usefulness of hyperlink structure for query-biased topic distillation. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and Development in information retrieval*, pages 48–455, Sheffield, UK, 2004.

[14] V. Plachouras, B. He, and I. Ounis. University of Glasgow at TREC2004: Experiments in Web, Robust and Terabyte tracks with Terrier. In *TREC '04: Proceedings of the 13th Text REtrieval Conference (TREC 2004)*, November 2004, NIST.

[15] V. Plachouras, I. Ounis, and G. Amati. The Static Absorbing Model for the Web. *Journal of Web Engineering*, 4(2):165–186, 2005.

[16] V. Plachouras, F. Cacheda, and I. Ounis. A Decision Mechanism for the Selective Combination of Evidence in Topic Distillation. *Information Retrieval*, 9(2):139–163, 2006.

[17] V. Plachouras. *Selective Web Information Retrieval*. PhD thesis, Department of Computing Science, University of Glasgow, 2006.

[18] J. M. Ponte and W. B. Croft. A language modelling approach to information retrieval. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281, Melbourne, Australia, 1998.

[19] J. Rocchio. Relevance feedback in information retrieval. In *The Smart Retrieval system - Experiments in Automatic Document Processing*, In Salton, G., Ed. Prentice-Hall Englewood Cliffs. NJ.313-323, 1971.

[20] I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, 1999.

Deploying Lucene on the Grid

Edgar Meij and Maarten de Rijke
ISLA, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam
The Netherlands
emeij,mdr@science.uva.nl

ABSTRACT

We investigate if and how open source retrieval engines can be deployed in a grid environment. When comparing grids to conventional distributed IR, the lack of a-priori knowledge about available nodes is one of the most significant differences. On top of that, it is also unknown when a particular node has time and resources available and starts a submitted job. Therefore, conventional methods such as RMI are not directly usable and we propose a different approach, using middleware designed specifically for grids. We describe GridLucene, an extension of the open source engine Lucene with grid-specific classes, based on this middleware. We report on an initial comparison between GridLucene and Lucene, and find a minor penalty (in terms of execution time) for grid-based indexing and a more serious penalty for grid-based retrieval.

The used middleware can gather a set of physical resources to form a single logical resource with some abstract properties. The user-definable properties can be used during indexing and retrieval to let GridLucene know which files it needs to access. By using this kind of semantic information, grid nodes can "discover" which indices exist on the grid and which particular documents need to be indexed.

GridLucene is available for downloading under the same license as Lucene.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Measurement, Performance, Experimentation

Keywords

Grid, grid computing, metadata, parallel indexing

OSIR 2006 August 6-11, 2006, Seattle, USA

1. INTRODUCTION

Grids are an emerging infrastructural technology for resource sharing. They enable the collaborative use of computational resources (commonly referred to as *nodes*), towards one or more common goals. Distributed or parallel computing is not a novelty, but there are significant differences when compared to grids. Within *clustered* computing, the individual components are tightly coupled, usually located near to each other and very homogeneous. Within a *distributed* approach, the homogenity is much less guaranteed, individual elements are loosely coupled and they can be geographically apart. Grids are like distributed systems, but they can cross national, as well as organizational boundaries. Grids also have interesting new properties that on the one hand ask for new solutions but can, on the other hand, also have interesting new potential.

The use of grids is still gaining in popularity and the enabling technologies are maturing as well. Increasingly, grids are being deployed in multi-site research projects and/or in projects where large amounts of data need to be stored or processed [31].

In this paper, we address the exploratory question if and how (open source) search engines can be deployed in a grid environment. Within our research group we make heavy use of Lucene [29], the well-known open source search engine, for various retrieval experiments and in various ways.[1] The performance of Lucene decreases with the ever-increasing size of document collections. How would it perform in a grid environment? What, if anything, is the difference between deploying an engine such as Lucene in a more conventional setup and deploying it in a grid environment? Are there any special challenges, benefits, or disadvantages? A significant difference between a grid-based approach and conventional distributed information retrieval for example, is the lack of a-priori knowledge about available nodes in a grid. This makes load balancing at runtime difficult, something which is still an ongoing research topic [1, 35]. Indeed, on the grid, it is usually not known in advance when a particular node

[1]We have used Lucene in a local fashion for example, with either a single index or with multiple indices for our Mood-Views project [27] and have also done experiments with a distributed setup. We have even augmented Lucene with an in-house developed Language Modeling extension [26] to replace the default similarity calculations. Faced with retrieval tasks that require substantial amounts of storage space, such as the TREC Terabyte retrieval track [10, 9], we turn to grids.

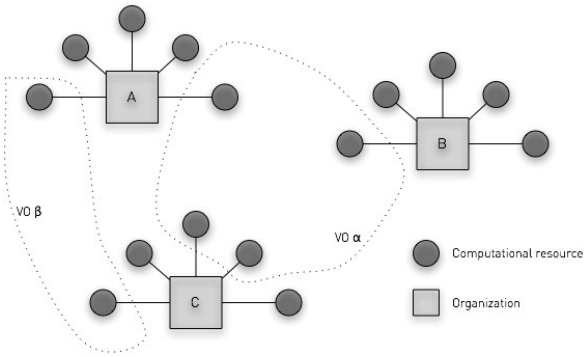


Figure 1: Organizations, sharing resources through Virtual Organizations.

starts a certain job, how many resources it has available, and where it is located—because of this, methods such as RMI (Remote Method Invocation) are not directly usable. To be able to take advantage of the distributed nature of grids, we propose a different approach, using grid-specific storage middleware. To evaluate this approach, we create a number of grid-specific implementations around Lucene.

The rest of this paper is organized as follows. Before we describe our approach in more detail, we first provide some background on grids (Section 2) and discuss several key components, common to most (institutional) grids, in Section 3. In Section 4 we describe the changes we made to Lucene and some additional implementations. In Section 5 we describe the setup used to evaluate indexing and searching performance of our grid-based solution. In particular, we compare indexing and retrieval speeds of our grid-enabled Lucene with a "traditional," locally running Lucene installation. The results can be found in Section 6, and concluding remarks are in Section 7.

2. BACKGROUND

A grid enables the integrated use of resources, which are typically owned by multiple organizations and/or individuals and is in fact a system consisting of distributed, but connected resources [12]. It also encompasses software and/or hardware that provides and manages logically seamless access to those resources [13, 24]. Grids can be roughly classified in two categories: institutional grids (IG's) and global computing or P2P (GCP) systems [3, 23, 11].

GCP systems typically harvest the computing power provided by individual computers, using otherwise unused bandwidth and computing cycles in order to run very large and distributed applications [22, 15]. Some examples include SETI@home [38], LookSmart's Grub [28] (a voluntary initiave to crawl the Internet in a distributed fashion), and ZetaGrid [40]. ZetaGrid is an attempt to verify Riemann's Hypothesis using grid technology, with a reported peak performance rate of around 7000 GFLOPS. There are also (open source) packages such as XtremWeb [11], and Q^2ADPZ [30] which allow to setup, deploy and run GCP projects. BOINC (the Berkeley Open Infrastructure for Network Computing) is another open source platform for public-resource distributed computing [3] and currently the enabling system for SETI@home, LHC@home, Einstein@home, Climateprediction.net, and many more.

Our work however, takes place in an IG environment. IGs cross organizational boundaries and are based on the virtual organization (VO) paradigm [20, 17, 21]. A VO is a dynamic collection of resources and users unified by a common goal and potentially spanning multiple administrative or organizational domains [18]. A figure depicting the notion of VOs is given in Figure 1 (adapted from a figure in [32]). IG's involve intensive computations and very large amounts of data, that also require secure resource sharing for which the current Internet infrastructure is unadequate. While GCP systems can be categorized as using weak to no authentication, IGs use strong authentication systems, based on the VO paradigm [19, 39]. They target broad scientific collaborations where various individuals, groups and institutions perform diverse calculations. There is considerable variation in the range of scientific grid applications, depending on the interest and scale of the community in question. A practical example of the use of an IG is the numerical solution of the long-open "nug30" quadratic optimization problem [33]. As opposed to mostly single machines within GCP systems, an IG resource might also be a cluster of machines, a storage system, database, or any other scientific instrument of considerable value that is administered in an organized fashion.

3. GRID COMPONENTS

In this section we describe the institutional grid environment in which we work and in which our experiments take place. We start out by describing the general environment and then zoom in on a particular kind of storage middleware and job submission details.

3.1 General

The construction of an institutional grid requires the establishment of standards for infrastructure, communication protocols and application programming interfaces [20, 37]. The Open Grid Services Architecture (OGSA) defines such a standard service-oriented IG framework [21]. The goal of OGSA is to standardize practically all the services one finds in an IG, such as VO/user management, security, resource management, job management, and data services. The implementation of these services provides a virtualization of a grid's resources and form an IG's middleware. Today, the Globus Toolkit [16, 25] is the de facto standard for any IG's middleware, with the Globus Toolkit 4 (GT4) being the most recent version [14]. GT4 is an open source toolkit and a realization of the OGSA requirements. It consists of services, programming libraries, and development tools which can be used to create IG-based applications [32].

Since there is no persistent storage on any grid node, specific middleware has been developed to accomodate storage requirements across an IG. Within GT4 there are several data management components such as GridFTP [20, 2] and OGSA-DAI (a service-based architecture for database access over the grid) [4]. Another implementation of a storage middleware solution will be presented in the next section.

3.2 Storage Resource Broker

The Storage Resource Broker (SRB) is a grid storage implementation, which includes data abstraction as well as metadata handling [5]. It provides a uniform API to heterogenous storage resources in a distributed fashion and has been used succesfully in various applications and fields, such as biomedicine, astronomy, digital libraries, and more [31].

SRB provides an abstraction layer over the actual storage devices, which can include various type of filesystems, as well as databases and archival storage systems. Because of this transparancy, SRB can accommodate virtually limitless amounts of data and various types of data scattered in multiple geographic locations can be accessed in a single representation from any grid node. Files and directories within SRB are organized in so called *collections*. With this abstraction, data items which are stored in a single collection can in fact be stored on heterogenous and geographically distant storage systems, but be referenced to by a single URI [5]. Individual files or directories can be accessed in the same manner. The available interfaces include a UNIX-like file I/O interface, a Java API called JARGON [8], and an interface that supports *get* and *put* operations of individual files, directories, or entire collections.

SRB also provides a metadata catalog service (MCAT), by which collections can be annotated, queried and retrieved, providing an abstract definition for resources. These annotations take the form of user-definable *attribute-value* pairs. They can be arbitrarily modified and queried, for example through JARGON [8]. This makes it possible to query SRB as one would a database management system, but instead of records you receive URI's of files or collections. Using these features, one can gather a set of distinct physical resources to form a single logical resource with some abstract properties.

3.3 Job Submission System

Unfortunately, GT4 was not yet functioning properly on the grid we are using for these experiments. We therefore used the tools made available through the European Data Grid project [6] to perform the current grid experiments with Lucene. These tools include a number of job planning, submission, and tracking tools (`edg-job-list-match`, `edg-job-submit`, and `edg-job-get-status` respectively), written in Python. Job descriptions are written in specialized configuration files, marked up using the Job Description Language.

4. A GRID-ENABLED LUCENE

To make Lucene run in our grid environment, we implemented a number of additions on top of the standard Lucene implementation. The main ones concern Lucene's interaction with SRB and the usage of MCAT, the metadata catalog service. The resulting implementation is called GridLucene; below, we briefly describe the additions that distinguish GridLucene from Lucene.

4.1 Storage

As described in subsection 3.2, each node on the grid has uniform access to files through storage middleware, such as SRB. Although it is possible, to some extent, to define on which particular machines or nodes our jobs will run, we decided not to make use of this functionality since it is not possible to define *when* our jobs will run. What we do know is that it is possible to describe which data from the storage middleware must be accessed and that this storage is available to all nodes through the network.

So, in order for Lucene to communicate with SRB, we have subclassed Lucene's (abstract) `Directory` class. This makes communications from Lucene with SRB transparent, because we can regard an SRB collection as any other local or RAM directory. Because of the implicit networking overhead, we wanted to evaluate the indexing and searching performance and compare it with a local, disk-based Lucene installation. The results of this evaluation can be found in Section 6.

4.2 Metadata

Using MCAT, the metadata catalog service within SRB introduced in Section 3.2, one can select certain files or collections from SRB, based on definable criteria. These criteria can be either more "regular" properties, such as filenames, sizes, and creation dates, but also manual annotations. For example, one can label a SRB collection and all the files therein `{collection:INEX, year:2005, indexed:no}`. A different set of files might have the same annotations, but a different value for one key: `year:2006`. When queried for `collection:INEX`, the SRB server will return the URI's of all the files in both collections. It would also be possible to just return the URI's of the collections themselves. Single files, directories, or collections can thus be located in a more semantic way than is normally possible using regular filesystems.

To make use of this new possibility, we have written Java classes which can partition a given document collection (consisting of multiple files) into subparts, put these on SRB, and annotate them. In a typical setting, one would split the collection into subsets, based on filesizes or the number of files, depending on the collection or task at hand.

This approach makes it possible for nodes to "discover" on their own which documents remain to be indexed. This means that, during indexing, each node that starts its job can select one of the remaining subparts of the document collection on the fly. It then indexes the documents in this subcollection and labels it as *done*. The indexing itself is preceded by transferring the actual documents from SRB and could be optimized by using either the local disk or RAM as a buffer. The index is written directly onto SRB or transferred afterwards from the buffer, so that it will be available in a later stage for retrieval. There are various possible approaches to transferring files from and to SRB, on which we elaborate in section 5.2.

The entire indexing process is shown graphically in Figure 2. Once the index has been stored on SRB, it is again possible to annotate it with specific metadata. It can, for example, be labeled with a description of its contents, the time taken to create the index, or the used `Analyzer`. Since the metadata is not restricted to pre-defined fields, the only limit is one's imagination or needs.

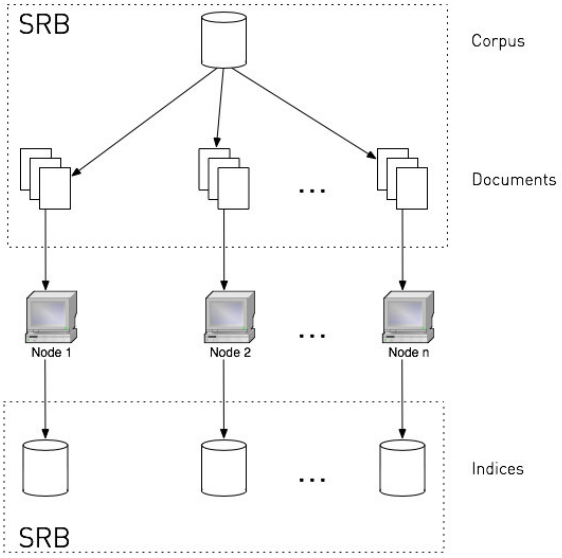


Figure 2: Indexing on a grid, using GridLucene. The document collection is split up into subsets. Each indexing node "discovers" which part(s) remain to be indexed based on SRB metadata. It selects a part, downloads and indexes the documents it contains into a separate index and transfers the index back to SRB. The index can then again be labeled with user definable metadata.

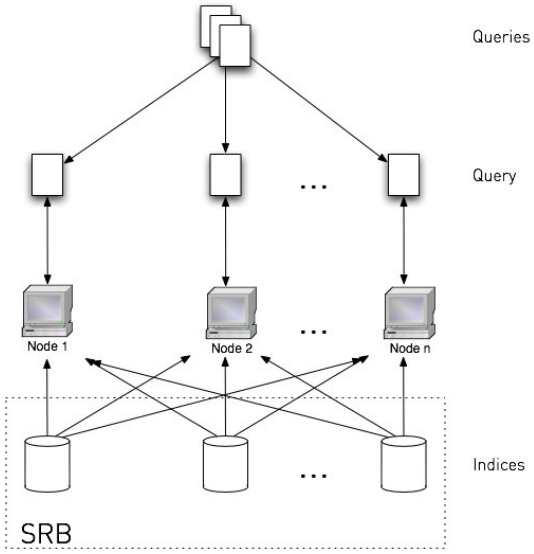


Figure 3: Searching on a grid. The separately created indices can be "discovered" by a searcher node, using SRB metadata. Each retrieval node then searches the appropriate index or indices.

At retrieval time, the searcher node(s) can query SRB to discover which directories have certain annotations and thus contain relevant indices for a certain query. They can then either load them into a Lucene `MultiSearcher` (if there is more than one relevant index), or a single `IndexSearcher` (If only a single index has been created, or if multiple indices have been merged into one). Both approaches can again make use of our SRB `Directory` class. In experimental information retrieval settings where very large batches of queries are more common, such as the TREC Terabyte track, it may also be beneficial to divide the queries into subsets and distribute these among the searcher nodes as depicted in Figure 3.

5. EXPERIMENTAL ENVIRONMENT

Now that we have described GridLucene, we turn to an evaluation of the efficiency of the additions implemented in GridLucene, and, more generally, of GridLucene.

We investigated the performance of the SRB-related classes, in order to assess response times and the scalability of the approach. Our goal is to see how Lucene performs in a grid setting, with varying document numbers to index and search, using the classes described in the previous section. Below, we present preliminary outcomes of "evaluations in progress." That is, at this stage our experiments do not yet provide an exhaustive benchmark analysis, but they do provide a sanity check in that they indicate how the performance of using SRB in combination with GridLucene relates to more conventional approaches.

5.1 Test Collection

For our experiments, we used the INEX 2005 test collection, which contains about 12,000 XML documents with an average size of 50 KB each [34]. We split it up randomly, based on filenumbers rather than sizes. In each experiment, we calculated the overall time taken using GridLucene and compared this with the results of Lucene running on a single machine (using the same Lucene version and settings, Java version, and document collection, but with everything stored locally on disk). This machine has dual Pentium 4 processors, each running at 3.0 GHz, 2GB of RAM, and Linux with kernel version 2.6.12 as operating system. We used Lucene version 1.9.1 and SRB version 3.4.1. Every indexing and searching run has been executed multiple (40), consecutive times, to compensate for any caching that may be involved; the times reported below are averaged over these 40 runs.

For indexing, we created document subcollections with an increasing numbers of files, starting with 1,000 documents and adding 1,000 at each increment, with an upper limit of 10,000 documents. The presented indexing times for GridLucene include transferring the documents from SRB and putting the generated index back on SRB, as described in Section 4.2. For searching, we used the indices generated during the indexing stage. GridLucene searches the indices stored on SRB, whereas the locally running Lucene uses locally stored indices. As queries we have taken 200 actual INEX topics[2] for this corpus and sequentially searched the index for each of them.

[2]`http://inex.is.informatik.uni-duisburg.de/2005/`

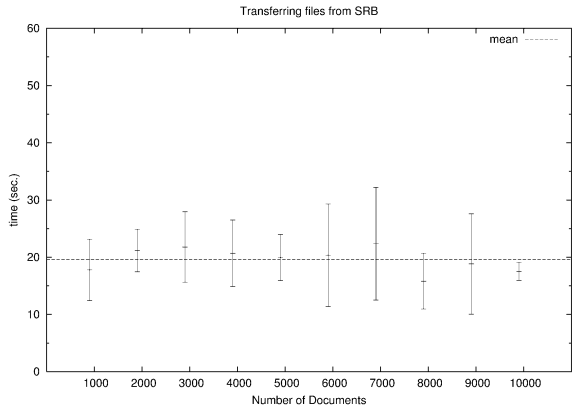


Figure 4: Time it takes to transfer an increasing number of files from SRB to a grid node. Due to the multi-threaded, parallel implementation, there is no noticable slowdown.

5.2 Transferring Files

We experimented with different ways of transferring the documents from and to a grid node. There are basically two distinct ways to transfer files from and to SRB [8]. The first is using methods from a Java `FileStream` like class, written to interact with SRB, which has roughly the same functionality as its Java counterpart. With its methods it is, for example, possible to seek a position in a file and read in an array of bytes. The other method is a higher-level class, which is able to transfer files using multiple threads. In turn, each thread can send multiple files in parallel, thus greatly improving performance.

For indexing we started out by using the `FileStream` approach to read in the source documents. This proved to be error-prone due to connection issues; a slight lapse in network connectvity caused the sequential stream to break unrecoverably. On top of that, this kind of transfer is also relatively slow. We therefore switch to a caching approach, in which we use the higher-level method to first download all the necessary documents to the indexing node. This does require a receiving node to have at least enough diskspace available to hold the files it needs to index. Figure 4 shows the average time it takes to transfer varying numbers of documents from SRB to a grid node, using the higher-level `copyTo` method. Further research is neccessary to determine how this method scales up beyond 10,000 documents and with varying document sizes. It seems that the 1 Gb/s ethernet connection is definitely not a bottleneck. When the index has been created it is transferred back to SRB using the same method. Before doing so, we first merge the resulting index segments into a single compound file.

We keep the original `FileStream` method on the retrieval side, since the searcher classes only need to access certain parts of the indexfile(s), namely those which contain the relevant parts of the inverted index and document mappings. We use the approach as described in section 4.1. For this evaluation we further use a single index, growing in size, and a single retrieval node. However, the implementation used can easily be extended to use multiple SRB indices in a Lucene `MultiSearcher`. We use the default retrieval model within Lucene.

6. EVALUATION

We wanted to know how long it takes GridLucene to index a certain set of documents from SRB to an SRB index and compare this with Lucene installed on a standalone machine. We also wanted to do a similar comparison for retrieval, as described in the previous section. The numbers in the figures presented in this section are the means of each run, with accompanying standard deviations.

6.1 Indexing

Figure 5a compares the total indexing time against an increasing number of documents. This also includes, in the case of GridLucene, the time to transfer documents from and indices to SRB. There is a slight variation in the transfer times of files from and to SRB, as can be seen from Figure 4. This variation also affects the results presented in figure 5a. It was clear from the beginning that GridLucene indexing times would be higher for any number of documents, due to the implicit networking overhead.

Although indexing using GridLucene is slower than a local setup, the extra time needed is limited and does not exceed 30 seconds. It also seems constant over every run, which may indicate that a trade-off point exists between conventional and grid-based indexing, at which the extra time needed for file I/O can be compensated by making subsets of the task and distributing these to multiple nodes. The additional indexing time involved when using GridLucene averages around 20 seconds per indexing task for these experiments, which may be an important predictor for this constant. Indexing 6,000 documents for example, would take around 60 seconds using a locally installed Lucene. Roughly the same amount of time is needed for GridLucene to index 2,000 documents. So, when dividing these 6,000 documents into three subsets and dispatching each to a grid node, the overall time taken is the same for both approaches. We have collected too little data however, to accurately determine if and where exactly the trade-off point lies and whether this would still hold for more and/or larger documents.

The fact that grid nodes running GridLucene can discover the annotated subcollections on their own is novel and can be seen as a step forward towards the acceptance of using grids within information retrieval.

6.2 Searching

Figure 5b shows a less optimistic picture. The searching times for GridLucene seem to increase more with a growing index size, than with a locally installed Lucene: file access method we currently use obviously fails for the task. We did not use the `copyTo` method from Section 5.2 for retrieval, thinking that it would not be necessary to copy the entire index to a searcher node. Instead, we held on to the FileStream approach, which clearly has its adverse effects on response times. Additionally, we did not perform any kind of local caching, so each time the index gets queried for a term, the request travels through the network. We had no means of quantitatively measuring the network overhead

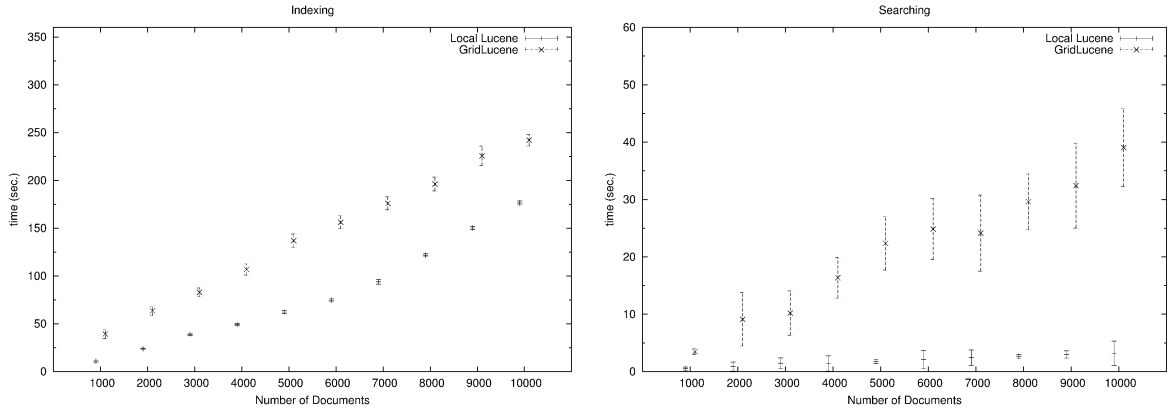


Figure 5: Indexing times of GridLucene (using SRB) compared with a local installation of Lucene and the time taken for 200 queries to be retrieved, with varying document numbers.

using the tools available on our grid, so we can only guess that caching the most-accessed parts of an index locally on a retrieval node will be beneficial.

Had we used the same method to transfer the index files as we did during indexing, the picture would probably have been similar to Figure 6a, with the response times of GridLucene being consistently higher (adding around 20 seconds) than the local Lucene. Whether this is in fact the case and to which extent this statement holds, remains a topic for further research.

7. CONCLUSIONS

We investigated if and how open source retrieval engines can be deployed in a grid environment. When comparing grids to conventional distributed information retrieval, the lack of a-priori knowledge about available nodes is one of the most significant differences. On top of that, it is also unknown when a particular node has time and resources available and starts a submitted job. Conventional methods such as RMI are therefore not directly usable and we propose a different approach, using middleware designed specifically for grids. As an example we have taken Lucene and implemented some grid-specific classes, based on this middleware. The resulting implementation, GridLucene, is available for downloading.

Some properties of the used grid storage middleware open up new and interesting possibilities. With relatively minor additions to Lucene we were able to transparently incorporate metadata about document (sub)collections and indices. Using user definable annotations, GridLucene can automatically find documents remaining to be indexed. The current approach thus makes it worthwhile to semi-automatically "parallelize" the indexing of large document collections.

Indexing new, additional documents in a later stage is also straightforward using a similar approach. They can be stored in a new index, or added to an existing one based on provided annotations. SRB also provides a theoretically infinite storage capacity, because additional storage space can be added without having to deal with changing file names and locations.

On the retrieval side, GridLucene can use the annotations to discover which indices exist on the grid and, most importantly, might be relevant. Especially in a grid setting, where sharing and collaboration are some of the main driving forces, these new possibilities give way to a whole new range of applications. In a sense, one can simply "plug in" additional indices during a search or discover what others might already have indexed. However, the way in which results from different information providers should be combined is still an ongoing issue within the distributed IR research field.

With GridLucene, a framework has been created in which research from the distributed information retrieval community can be tested in a grid setting. For example, the resource *description*, *selection* and *merging* algorithms for distributed/federated text databases [7] can be more or less directly applied in a multi-organizational grid environment.

We carried out some initial benchmarkings on the proposed implementations and found that GridLucene incurs a minor and consistent penalty when indexing on SRB, while the online performance during document retrieval incurs a far more dramatic penalty. This is most likely due to the chosen file-access methods—while appropriate in a grid-based environment, it proves to be a relatively slow and error-prone file access method in the current setting. It seems there is a trade-off factor between the performance of distributing the indexing task among grid nodes (using GridLucene) and local indexing. We did not collect enough data to test whether this is in fact the case and where it plateaus. This is therefore something we intend to study further, for example using larger and more heterogeneous corpora

We have not yet investigated whether a lower level alteration of Lucene might also be fruitful in a grid environment. When GT4 becomes available on our grid, we intend to answer that question as well. Furthermore, we plan to use and evaluate

divide-and-conquer grid API's such as IBIS [36, 35] in an information retrieval setting.

8. AVAILABILITY

GridLucene is available for download at `http://ilps.science.uva.nl/Resources/`, under the same license as Lucene is distributed with.

9. ACKNOWLEDGMENTS

Thanks to Gilad Mishne for his insights into the inner workings of Lucene, to Machiel Jansen for his last minute comments, and to the reviewers for their constructive comments.

This work was carried out in the context of the Virtual Laboratory for e-Science project (`http://www.vl-e.nl`). This project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ).

Maarten de Rijke was supported by the Netherlands Organisation for Scientific Research (NWO) under project numbers 017.001.190, 220-80-001, 264-70-050, 354-20-005, 612-13-001, 612.000.106, 612.000.207, 612.066.302, 612.069.006, and 640.001.501.

10. REFERENCES

[1] M. Aldinucci, M. Coppola, S. Campa, M. Danelutto, M. Vanneschi, and C. Zoccolo. Structured implementation of component based grid programming environments. In *Dagstuhl Seminar Future Generation Grid 2004*, CoreGRID series. Springer Verlag, 2005.

[2] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, and I. Foster. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *MSS '01: Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, page 13, Washington, DC, USA, 2001. IEEE Computer Society.

[3] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[4] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. P. C. Hong, B. Collins, N. Hardman, A. C. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N. W. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead. The design and implementation of grid database services in ogsa-dai. *Concurrency and Computation: Practice and Experience*, 17(2–4):357–376, 2005.

[5] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Procs. of CASCON '98*, Toronto, Canada, 1998.

[6] R. Berlich, M. Kunze, and K. Schwarz. Grid computing in europe: from research to deployment. In *CRPIT '44: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 21–27, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[7] J. Callan. Distributed information retrieval. In W. B. Croft, editor, *Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval*, The Kluwer International Series in Information Retrieval, pages 127–150. Kluwer Academic Publishers, 2000.

[8] S. D. S. Center. JARGON, a Java API for the DataGrid, 2006. `http://www.sdsc.edu/srb/jargon`.

[9] C. Clarke, N. Craswell, and I. Soboroff. The TREC Terabyte retrieval track. *SIGIR Forum*, 39(1):25–25, 2005.

[10] C. Clarke, F. Scholer, and I. Soboroff. The TREC 2005 Terabyte track. In *The Fourteenth Text REtrieval Conference (TREC 2005) Proceedings*, 2005.

[11] G. Fedak, C. Germain-Renaud, V. Neri, and F. Cappello. XtremWeb: A generic global computing system. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 582, Washington, DC, USA, 2001. IEEE Computer Society.

[12] I. Foster. Internet Computing and the Emerging Grid. *Nature*, 7 Dec. 2000.

[13] I. Foster. What is the grid? A three point checklist. *GRIDtoday*, 1(6), 2002. `http://www.gridtoday.com/02/0722/100136.html`.

[14] I. Foster. Globus Toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer Verlag, 2005.

[15] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, Feb. 2003.

[16] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[17] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[18] I. Foster, C. Kesselman, L. Pearlman, S. Tuecke, and V. Welch. The community authorization service: Status and future. In *Proceedings of Computing in High Energy Physics 03 (CHEP '03)*, 2003.

[19] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[20] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.

[21] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The Open Grid Services Architecture, Version 1.0. Technical report, Global Grid Forum, 2005. `http://www.ggf.org/documents/GFD.30.pdf`.

[22] C. Germain-Renaud, G. Fedak, V. Néri, and F. Cappello. Global computing systems. In *LSSC '01: Proceedings of the Third International Conference on Large-Scale Scientific Computing-Revised Papers*, pages 218–227, London, UK, 2001. Springer-Verlag.

[23] C. Germain-Renaud and D. Monnier-Ragaigne. Grid result checking. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 87–96, New York, NY, USA, 2005. ACM Press.

[24] GGF. Global Grid Forum, 2006. `http://www.ggf.org`.

[25] Globus. The Globus toolkit, 2006. `http://www.globus.org/toolkit`.

[26] ILPS. The ILPS extension of the Lucene search engine. `http://ilps.science.uva.nl/Resources`.

[27] ILPS. Moodviews. `http://www.moodviews.com`.

[28] LookSmart. Grub's distributed web crawling project, 2006. `http://grub.looksmart.com`.

[29] Lucene. The Lucene search engine. `http://lucene.apache.org/`.

[30] Q^2ADPZ. `http://qadpz.idi.ntnu.no`.

[31] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.-Y. Chen, and R. Olschanowsky. Storage resource broker - managing distributed data in a grid. In *Computer Society of India Journal, Special Issue on SAN*, volume 33, pages 42–54, October 2003.

[32] B. Sotomayo and L. Childers. *Globus Toolkit 4: Programming Java Services*. The Morgan Kaufmann Series in Networking. Morgan Kaufmann, 2006.

[33] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. In *Concurrency and Computation: Practice and Experience*, 2004.

[34] A. Trotman and M. Lalmas. Introduction to the inex 2005 workshop on element retrieval methodology. In *Proceedings of the INEX 2005 Workshop on Element Retrieval Methodology, Second Edition*, 2005.

[35] R. V. van Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, and H. E. Bal. Ibis: an efficient Java-based grid programming environment. In *Joint ACM Java Grande - ISCOPE 2002 Conference*, pages 18–27, Seattle, Washington, USA, November 2002.

[36] K. van Reeuwijk, R. V. van Niewpoort, and H. E. Bal. Developing Java grid applications with Ibis. In *Proc. of the 11th International Euro-Par Conference*, pages 411–420, Lisbon, Portugal, September 2005.

[37] G. von Laszewski, P. Z. andTan Trieu, and D. Angulo. The Java CoG kit experiment manager. Technical report, Argonne National Laboratories, 2006.

[38] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela. SETI@HOME – Massively distributed computing for seti. *Comput. Sci. Eng.*, 3(1):78–83, 2001.

[39] L. J. Winton. A simple virtual organisation model and practical implementation. In *CRPIT '44: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 57–65, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[40] ZetaGrid. `http://www.zetagrid.net`.

IR-Wire: A Research Tool for P2P Information Retrieval

Shefali Sharma, Linh Thai Nguyen, Dongmei Jia
Illinois Institute of Technology
Chicago, IL 60616, USA
{sharshe, nguylin, jiadong}@iit.edu

ABSTRACT
We introduce a system for information retrieval research in the peer-to-peer file-sharing domain. Our system, IR-Wire, is based on the popular Gnutella protocol, giving us access to a large user base and a large data set. As a search tool, IR-Wire maintains many statistics and implements a number of information retrieval ranking functions. As a research tool, our main focus, IR-Wire contains a data logger and analyzer. The data logger logs both incoming and outgoing queries and query results and provides a way to create a snapshot of the entire data set shared by the users. The data analyzer provides a simple user interface for data analysis. We briefly discuss an analysis conducted on a million incoming queries that were collected in our log files.

Categories and Subject Descriptors
H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Web-based services.*

General Terms
Algorithms, Design, Performance.

Keywords
IR-Wire, information retrieval, open source, query log analysis.

1. INTRODUCTION
Peer-to-peer (P2P) technology [6] is very popular, with file sharing being a leading P2P application. With the demise of Napster in July 2001 both semi-centralized systems like FastTrack and completely decentralized P2P file sharing systems like Gnutella gained popularity. Gnutella is the third-most-popular file sharing network in the Internet. It is thought to host on average approximately 2.2 million users daily, although around 400,000 to 500,000 are online at any given moment [8]. Because of the immense popularity of these networks, it is imperative to find ways to improve their performance. Research tools for easy experimentation are needed that will help study both the data flowing through these networks and the behavior and preferences of their users. Although such tools exist for centralized systems (Section 2), we know of no readily available P2P counterparts.

We develop a system built on top of LimeWire's Gnutella system [9]. LimeWire is an open source program written in Java. It allows users to share any type of file and runs on Windows, Macintosh, Linux, Sun, and other computing platforms. Specifically, to the LimeWire client we have added the following functionalities:

- Statistics maintenance – Each client collects statistics about shared files.
- Information retrieval-style ranking functions – Each client implements a number of ranking functions.
- Data Logger – Each client can log incoming data, and outgoing data with results.
- Data Analyzer – An independent component that analyzes logged data.

2. RELATED WORK
Much research on peer-to-peer systems focus on how to model the peer-to-peer network [16-19]. Most of them deal with modeling the lower levels of peer-to-peer network, such as modeling peer-to-peer protocols, modeling the way files are replicated over the network, or modeling the network topologies. Our work does not focus on the lower level of the network, but the application level of a peer-to-peer information retrieval system.

There are a number of peer-to-peer information retrieval projects that have been developed, such as Peerware [12], Anthill [20], Alvis [21], and Nutch [22], to name a few. Most of these projects focus on query routing [12], resource discovery [20], or the design structure of a peer-to-peer information retrieval system [21]. Most of them implement search capabilities, but have less of a focus on data logging and analysis.

Most of current works on query log analysis are done for Web search engines. Silverstein et al. [1] analyzes a very large collection of queries logged by AltaVista for a period of six weeks, it contains almost one billion queries. Beitzel et al. [2] analyzed the changes of queries in terms of query popularity and uniqueness over time.

Other works are similar, in that they also reported statistics about query length, query distribution, the most popular queries and the most popular terms [3-5]. Some works also reported user behavior like the length of the query entered or how often the initial queries are modified.

To our knowledge, little work has been done on query logging as well as query log analysis for peer-to-peer systems even though researchers in the field have agreed that one is required [23]. The reason may be that it is assumed that queries in peer-to-peer systems are similar to queries in Web search engines, thus observations from Web search engines query logs can also be used to model peer-to-peer systems.

Zeinalipour-Yazti and Folias logged query messages and analyzed them for Gnutella network in 2002 [14]. Their work is most similar to our work. To collect user queries, they have deployed

their Peerware on 17 workstations to collect all pass-through-messages. In a period of 5 hours, they have collected about 15 million query messages. The limitation of their approach is that, not all query messages are forwarded to their peers. Most likely, the query messages they have received are similar to what they were sharing. Thus their query collection may not be representative of all queries flowing through the network. In developing IR-Wire we have modified our routing tables to capture all queries flowing through the network. Therefore, we receive a complete set of all user queries in a given time period. In addition, Zeinalipour-Yazti and Folias collected peer-to-peer queries in June 2002. Therefore, their query collection may be out-dated, since what the users were looking for 4 years ago may be totally different from what they are looking for today.

IR-Wire helps build an unbiased query collection. It focuses on IR research in a large-scale Gnutella-based peer-to-peer, file-sharing application allowing users to collect more up-to-date queries and providing an easy interface for doing analysis on the data set.

3. BRIEF DESCRIPTION OF GNUTELLA

We choose to base our work on the Gnutella protocol because it is not just popular but also well studied. In the Gnutella network, a user searches for a file by issuing a keyword query. This query is sent to all the clients it is actively connected to (this number is usually small, on the order of 10). The client further forwards this query to all of its neighbors in the Gnutella network. This process repeats until the query's Time-To-Live (TTL) expires or the packet has reached a client that is a predetermined number of "hops" away from the sender [7].

Gnutella specifies an unstructured and highly distributed network where each node is fully autonomous, independently controlling its local repository of shared files. Incoming queries are compared against the files descriptors (since these shared files are binary files, they need external descriptors). File descriptors are generally implemented via file names. Matching results are returned to the peer that issued the query.

4. SYSTEM DESCRIPTION

The goals of IR-Wire are twofold: a search tool and a research tool. The overall architecture of our system is now described along with an explanation of the functionality of each component.

4.1 IR-Wire System Architecture

The architecture of the system is depicted in Figure 1. IR-Wire's logging functionalities can be turned off if basic LimeWire functionalities are desired. The Data Analyzer is separate from the "main" LimeWire system. Loading the logged data into a MySQL database is an optional, separate batch process.

4.2 System Components

Our system consists of the following modules: *IR+* for improving search; the LimeWire system with few modifications; Data Logger, Loader and Analyzer modules.

The modifications made to the core LimeWire system is minimal, which should simplify the incorporation of our enhancements into future versions of LimeWire.

4.2.1 IR+

This module uses the meta-data associated with the file and utilizes IR techniques to rank results, disambiguating them, thereby improving the quality of the results for a particular search. It implements the following additional ranking functions [10]:

- Term Frequency: The result whose descriptor contains the most query terms is ranked highest.
- Fraction: The result whose descriptor covers the highest fraction of query terms is ranked highest.
- Cosine Similarity: A result is ranked based on the cosine similarity score between the query and its descriptor.
- TF/IDF: This is similar to cosine similarity, but with each term weighted based on document frequency. This ranking function requires an estimation of the global document frequency of each term computed using the local shared repository.
- Group size: Results that refer to the same file are grouped and the rank score is the size of the group. This ranking function is implemented in most P2P file-sharing systems, including LimeWire.

This list of ranking functions is not exclusive. The IR+ component is designed such that the implementation of additional ranking functions is straightforward.

4.2.2 LimeWire System-Query Logging

The original Gnutella protocol floods queries in the network, limiting its scalability [7][25]. To deal with the scalability concern, *ultrapeers* [19] are used. An ultrapeer is a peer that is judged to be highly reliable and capable of handling more Gnutella workload. The set of ultrapeers makes possible a two-tier system, with ultrapeers and *leaf* (regular) nodes. Ultrapeers flood queries to each other, and leaves connect only to ultrapeers (not to other leaves). Ultrapeers can be seen as "proxies" for leaf nodes in the Gnutella network.

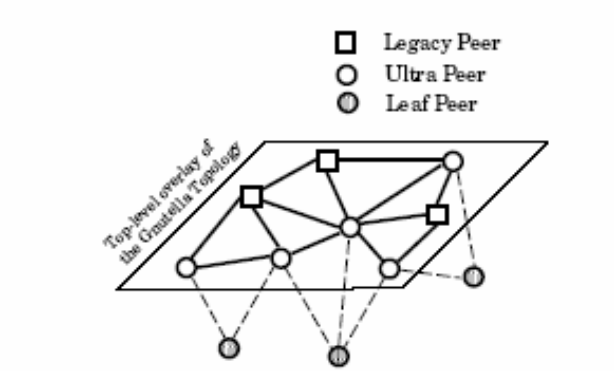


Figure 1. Two-Tier Topology of Modern Gnutella

Leaf nodes interact with ultrapeers to control the amount of incoming queries by the Query Routing Protocol (QRP) [11]. The QRP specifies that nodes create query routing tables by hashing keywords of all the file descriptors that they share and storing these hash values in a bit-vector (a form of the Bloom Filter [11]). By exchanging these routing tables with neighbors, nodes know what queries their neighbors can match and make routing decisions appropriately. Leaf nodes create these routing tables, and send them to their ultrapeers. Ultrapeers, thus, route only a subset of the incoming queries, the queries that are likely being answered, to their leaves.

Specifically, queries are conjunctive – a query matches a descriptor if all query terms are contained in the descriptor. A bit-vector (i.e., routing table) with more bits set, therefore, receives more queries. In practice, bits are set as files are added to the shared local repository. Clients with few shared files, therefore, have very sparse bit-vectors.

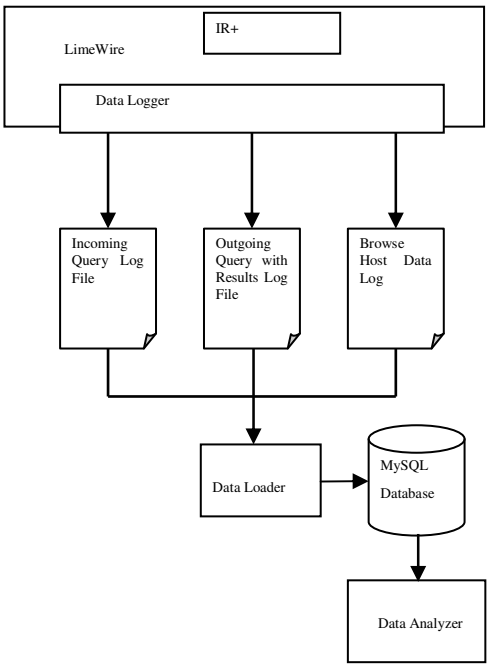


Figure 2. Architectural Design of IR-Wire

Our goal is to log all queries that are being issued in the Gnutella network to get an accurate picture of what users of peer-to-peer file sharing systems are searching for. One way to do this is to share as many files as possible. This option is not realistic, given the size of the bit-vector (64KB [11]). Another option is to make our LimeWire node an ultra-peer. This option also, is not practical, as it requires either maintaining a high bandwidth Gnutella node for a long period of time, thereby being elected as an ultrapeer, or modifying the LimeWire software to allow a client to masquerade as an ultrapeer. The former is difficult because it requires long-term dedicated resources, without any guarantees that the client will become an ultrapeer. The latter, masquerading as an ultrapeer, is possible, but introduces instability into the network if our ultrapeer frequently joins and leaves.

To get all incoming queries for our analysis, we choose an easier option. We set all the bits in the bit-vector to falsely claim that we are sharing files that contain all the keywords the users are searching for. This causes the ultrapeer's bit-vector to be set with all ones as well, causing all the network traffic to be routed to that ultrapeer, and then, of course, to us. This may cause some flooding in the network but is necessary in yielding an unbiased data set.

4.2.3 Data Logger and Analysis Component

This component is responsible for logging and analysis of all the different types of queries with the results obtained from the search. It contains the following sub-components:

- The log files - IR-Wire maintains three types of log files. Other than storing the incoming queries in the incoming query log file, IR-Wire also allows the user to log outgoing queries as well as the query's results in another log file. Work is also in progress to facilitate storing the data retrieved from browsing of peers' shared repositories. LimeWire allows users to view the contents of a peer through the "browse host" facility. IR-Wire will use this facility to gather data on a host, and then use this data to perform other searches and other browse host actions. It will thus create a snapshot of the data shared by all the hosts.
- Data Logger - The Data Logger reads from and writes to the log files. Data Logger functions are called from within the LimeWire code to log data to the appropriate log files or view the log through the LimeWire interface.
- Data Loader – The Data Loader reads the log files and stores the data in the corresponding MySQL tables. It has no interaction with the LimeWire code. Loading is done as a batch process if the user wants to use the tables.
- Data Analyzer – The Data Analyzer allows the user to perform various analyses through a simple user interface where selections can be made. The interface is under construction.The user will be able to either write SQL queries to the database and save the results retrieved or select from the analysis already implemented. The analyses that have already been implemented are for the incoming query messages[1]. These include retrieving average query length, query length distribution, queries popularity distribution, query categorization by the type of file (audio/video/program) desired by the user, and correlation analysis (e.g., those performed in [1]). We have discussed some results of these analyses in Section 5. This module also does not interact with the LimeWire code. Analyses on the data are performed on the log files or querying is done on the tables depending on the user preference.

4.3 Limitations to our Logging Capabilities

Our current version of IR-Wire is limited in its data logging capabilities due to the specifics of the Gnutella protocol. These specifics do not allow us to collect specific "session data" on users, which prevents session analysis (e.g., how the user modifies his/her queries to improve the results received),

Gnutella query routing is done on a hop-to-hop basis. Queries are routed from a peer to its neighbors, who are not given information on where the peer got the query from in the first place. Consequently, query replies (which do contain the IP address of the replying node) must follow a reverse path to arrive at the query originator.

[1] These are the incoming query requests or simply query messages. The standard Gnutella protocol contains five types of messages: ping (discover hosts on network), pong (reply to ping), query (search for file), query hit (reply to query), push (download requested for firewalled servents).

Reverse path routing is likely a security feature [24], increasing the anonymity of users. We are currently seeking holes in this security mechanism that may allow us to track users.

Another limitation is that we were not able to get the time the query was sent by the user. This and the fact that we cannot identify the originators of queries prevent us from doing session analysis which would help us to study user behavior.

5. QUERY LOG ANALYSIS

We analyze the attributes of the incoming query log and report our findings. The analysis here is cursory, and meant only to demonstrate what kind of analyses are possible with the data collected by IR-Wire.

5.1 LimeWire Search Engine

LimeWire allows users to share files of any type, like .mp3, .avis, .jpgs, .tiffs etc. It is capable of multiple simultaneous searches, available in several different languages. Also, it allows users to make many different types of searches. Other than the normal keyword based queries, users can also search for the shared data on a given IP address (the "browse host" facility). Even for the keyword based queries the user can query using just the query string or he/she can specify other attributes like genre, album, or artist. This type of query is called a "rich query." In the keyword based search the user can have either an un-constrained search which searches for files of any type or a constrained search for a specific type of file like audio, video, program, etc.

In our analysis, we have restricted ourselves to keyword based queries that are not rich queries. We are also restricting ourselves to only English queries (LimeWire allows queries to me made in many different languages). As we will show, these restrictions should not significantly hurt the generality of our results, as most queries are keyword-based, English queries.

5.2 IR-Wire Incoming Query Log

We kept our IR-Wire running the whole day to collect incoming queries on Saturday, May 13th, 2006. Over the weekend, there is less business traffic and more recreational traffic on the network, compared with the case of weekdays. Since Gnutella is arguably used primarily for recreational purposes, we should be able to yield a reasonably representative set of queries on the weekend. We collected the data in 20 log files of 100MB each. Each line in the log file is a query request. The following attributes are stored for each query: The original query string, the time when the query was received, values to indicate if the search was constrained or un-constrained and type of file the user is searching for (e.g., audio, video, program, document, image).

We preprocess all the query strings by ignoring any case differences, removing stop words, replacing any punctuation with white space, and compressing white space to single spaces. Furthermore, we observed that some queries include unrecognizable non-ASCII strings. We believe that most of them are non-English queries, which are removed from the analysis. This constituted about 25% of the query log.

After preprocessing, totally there are 775,605 queries and 65% (498,130) of them are distinct queries without considering replicas[2]. Note that we ignore the capital letter in our analyses.

[2] The collected query logs are available upon request.

5.3 Initial Findings

As shown in Table 1, the average number of terms per query is 2.94, which validates that queries tend to be short in general. Similar average query length is reported in [5], which indicates that a short query trend exists in both peer-to-peer file-sharing systems and centralized Web search engines.

Table 1. Statistics on Query Length and Query Frequency

	Query Length	Query Frequency
Max	12	623
Min	1	1
Average	2.94	1.56
Std Dev	1.27	2.21

Besides that, we report the query length distribution in Figure 4. Only unique queries are considered in this case. Most of the queries contain 1 to 6 query terms. There is only one query that has 10, 11 or 12 terms and 3 queries which have length 9. 75% of queries contain 2 to 4 keywords.

In Figure 5, we compare the frequencies of unique queries with their frequency ranks. The query ranked 1st is the most popular (frequent) query and next one is the 2nd most popular query and so on. Queries that have exactly the same query terms are treated as replicas of one query, even if clients type in the query terms in a different order. We observe that query popularity follows a Zipf-like distribution. In contrast to the fact that the frequency of the most popular query is 623, about 75% of all the queries occur only once in the whole day, which indicates that the interests of Gnutella users are quite diverse or users tend to describe their needs differently. Later we show what exactly those top ranked queries are.

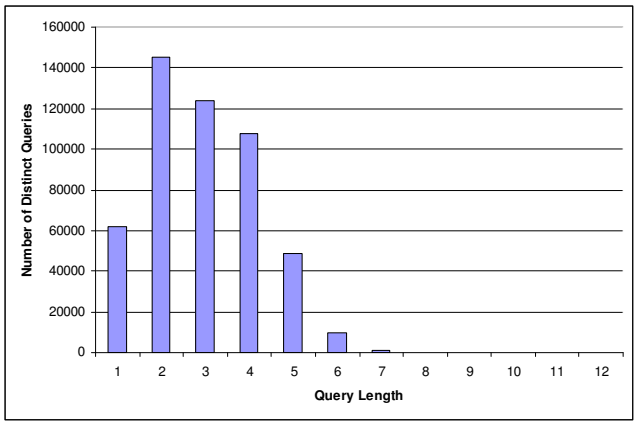


Figure 4. Query Length Distribution

We categorized all the logged queries based on the desired types included in a query request message. As shown in Figure 6, a large portion of queries does not have constraints on the desired

file types, which is represented by the legend "all". In the cases that one or multiple types are defined by clients along with keywords, more than 70% of the queries are for audio files, 20% are for video and the other 10% queries are for images, programs and documents. This is the same as we expected, because most clients in a P2P file-sharing system tend to search for songs or movies on the purpose of entertainment, other than documents or programs.

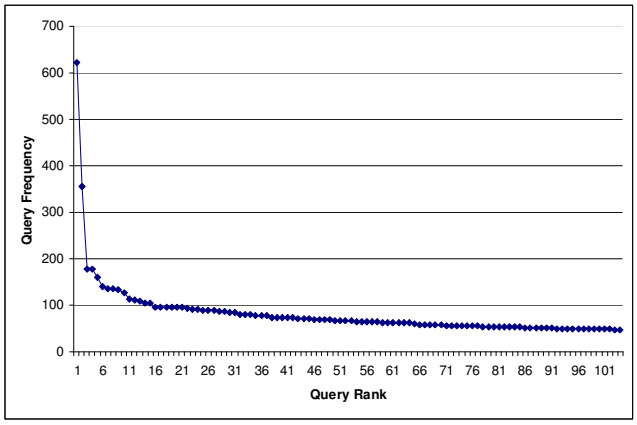


Figure 5. Query Popularity Distribution

In Table 2, we report exactly the 25 most popular queries with their frequencies and the top 25 keywords with the number of queries in which each keyword is contained. Surprisingly, the query "neonode" is the most popular query. As far as we know, Neonode is a Swedish manufacturer of mobile phones [15]. Other mysterious queries occur as well, such as "pdmckaziejdntb." Because a search for "neonode" using IR-Wire returns 0 byte files, we are guessing that neonode may be a "control query," used for tracking network connectivity.

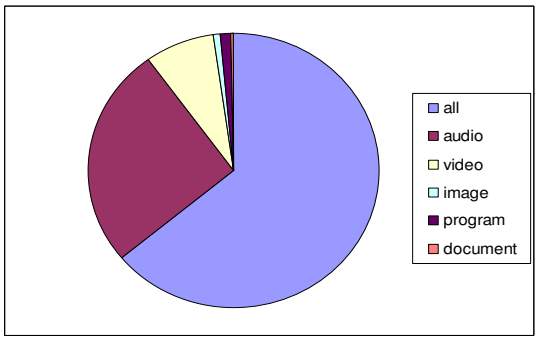


Figure 6. Query Categorization by Types of Desired Files

One interesting observation is that a couple of the top 20 queries, such as "poseidon," "mission impossible," "da vinci code" and so on, are newly released movies. So they are in high demand and have been searched frequently by Gnutella clients during the time period of our testing. It indicates that query popularity in P2P file-sharing systems is closely related to the temporal aspect. Hence, it is not necessary that a popular query in the current time period is popular consistently over time. This can be examined in greater detail by measuring the query frequency distribution and recording the most popular queries periodically at various weeks or even months, which is left as our future work.

We also notice that some file extensions, avi, mp3 and dvd, are among the most popular query terms. The reason we believe is that for an original query containing a keyword such as song.mp3, it becomes two terms song and mp3 after replacing all the punctuations by white space which is used to detect and separate two adjacent keywords in the preprocessing phase. Due to the large amount of queries for music or movies, these terms occurs much more times and are counted as popular query terms.

Table 2. Top 25 Most Popular Queries and Terms (After Preprocessing)

Top 25 queries	Freq	Top 25 terms	Num Queries Contained
Neonode	623	love	6235
Love	355	dj	5172
Pdmckaziejdntb	178	los	4926
Poseidon	160	live	4790
Pthc	139	feat	3822
Time	135	avi	3737
smallville vessel	135	ft	3668
bibcam	134	big	3627
ptsc	126	remix	3486
lsbar 001a kdquality	114	girl	3289
divx ita men dvd rip	112	im	3255
mission impossible	108	sex	3253
99bb	105	mp3	3135
lco	96	star	3090
istock	95	pthc	3088
civilization 4 iv	95	john	2921
kokeshi	95	te	2873
afford www buylegalmp3	95	time	2860
istock l2	95	en	2821
mihimaru gt	93	xvid	2802
eureka	92	day	2700
da vinci code tom hamks	91	full	2661
fm2006	90	dvd	2652
lsh	89	man	2618
template zip	88	life	2611

6. CONCLUSION AND FUTURE PLANS

In this paper we introduced a P2P file-sharing system called IR-Wire built on top of Limewire's Gnutella system to monitor and

collect user queries, in order to reveal the real-world of queries in Gnutella network. This work is meant to address a need for research tools and data for P2P IR, expressed in [23]. We also presented our observations and analysis of almost 1 million incoming queries collected in the log. The statistical results show that queries tend to be short. 75% of queries contain 2 to 4 keywords. Query popularity follows a Zipf-like distribution and the majority of queries are for music and movies. A lot of query terms are closely related to temporal aspect, so their popularity may shift dramatically over time.

Various other analyses will be implemented for the incoming queries, new analyses will be done on outgoing queries and the results returned. Among other things in the pipeline, there is a logging and analysis of the "browse host" queries which will provide a good insight into the data users are sharing.

For the incomming queries, we plan to use a data mining technique (association rule mining) to reveal the correlation among query terms, the correlation among attribute-values of the rich queries, as well as the correlation between a query term and an attribute-value. This information, if available, is very helpful, since it can be used to optimize the query routing protocol of peer-to-peer systems.

We also plan to collect the results returned by the system for each user query. One approach is to use the logged incoming queries as our node's outgoing queries and record all returned results. The use of automatically generated queries is also possible. The set of returned results for each query may give us some knowledge about how efficient the search in peer-to-peer network is.

We also want to investigate whether there is a correlation between the shared collection of a user and the set of queries s/he issued. Our conjecture is that, users are normally looking for files that are similar to those they are sharing.

The code and the data collected will be available for download from the IIT IR website (http://www.ir.iit.edu), or by request.

7. ACKNOWLEDGMENTS

We would like to thank the reviewers of this paper for their very helpful comments.

8. REFERENCES

[1] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. *Analysis of a Very Large Web Search Engine Query Log*. SIGIR Forum, 33(1):6--12, 1999.

[2] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder. *Hourly Analysis of a Very Large Topically Categorized Web Query Log*. SIGIR'04, 321-328, 2004.

[3] B. J. Jansen, A. Spink, and J. Pedersen. *An Analysis of Multimedia Searching on AltaVista.* MIR'03, 186-192, 2003.

[4] A. Broder. *A Taxonomy of Web Search.* SIGIR Forum, 36(2), Fall, 2002.

[5] A. Spink, S. Ozmutlu, H. C. Ozmutlu, and B. J. Jansen. *U.S. Versus European Web Searching Trends.* SIGIR Forum 36(2), 32-38, 2002.

[6] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja1, J. Pruyne, B. Richard, S. Rollins, Z. Xu. *Peer-to-Peer Computing*. HP Laboratories Palo Alto, HPL-2002-57 (R.1), July 3rd , 2003

[7] LimeWire Technical Document. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf

[8] http://www.slyck.com/news.php?story=814

[9] LimeWire home page. http://www.limewire.com

[10] W.G. Yee, O. Frieder. The Design of PIRS, a Peer-to-peer Information Retrieval System, DBISP2P 2004 Workshop

[11] http://rfc-gnutella.sourceforge.net/src/qrp.html

[12] *Peerware,A Real P2P Information Retrieval Testbed.* Web document, http://www.cs.ucr.edu/~csyiazti/peerware.html

[13] LimeWire Technical Document. http://www.limewire.com/developer/Ultrapeers.html#_ftn1

[14] D.Zeinalipour-Yazti, and T. Folias. *A Quantitative Analysis of the Gnutella Network Traffic.* TR-CS-89, Dept. of Computer Science, Univ. of California, Riverside, 2002

[15] http://en.wikipedia.org/wiki/Neonode

[16] L. Zou, and M. H.Ammar. *A File-Centric Model for Peer-to-Peer File Sharing Systems.* Proc of the 11th IEEE Intl. Conf. on Network Protocols (ICNP'03), 2003.

[17] S. Merugu, S. Srinivasan, E. Zegura. *P-sim: A Simulator for Peer-to-Peer Networks*. Proc. of the 11th IEEE Intl. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'03), 2003.

[18] W. Yang, N. Abu-Ghazaleh. *GPS: A General Peer-to-Peer Simulator and its Use for Modeling BitTorrent*. Proc of the 13th IEEE Intl. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, (MASCOTS'05), 2005.

[19] Q. He, M. Ammar, G. Riley, H. Raj, and R. Fujimoto, *Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems*. Proc. of the 11th IEEE Intl. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'03), 2003.

[20] O. Babaoglu, H. Meling, and A. Montresor. *Anthill: A Framework for the Development of Agent-based Peer-to-Peer Systems.* Proc. of the 22nd Intl. Conf. on Distributed Computing Systems (ICDCS'02), July 2002.

[21] K. Aberer, F. Klemm, M. Rajman, and J. Wu. *An Architecture for Peer-to-Peer Information Retrieval.* Proc. of the 7th Annual Intl. ACM SIGIR Conf. Wrkshp on Peer-to-Peer Information Retrieval, July 2004.

[22] Nutch project home page. http://lucene.apache.org/nutch/

[23] H. Nottelmann, K. Aberer, J. Callan, and W. Nejdl, CIKM 2005 P2PIR Workshop Report, 2005, http://p2pir.is.informatik.uni-duisburg.de/2005/report.pdf

[24] D. Bickson, D. Malkhi, A Study of Privacy in File Sharing Networks.

[25] J. Ritter, Why Gnutella Can't Scale. No, Really., Web Document, February, 2001, www.darkridge.com/~jpr5/doc/gnutella.html.

Tagging in Peer-to-Peer Wikipedia

A Method to Induce Cooperation

Jenneke Fokker
Dept. of Industrial Design
Delft University of Technology
The Netherlands
j.e.fokker@tudelft.nl

Wray Buntine
Dept. of Computer Science
Helsinki Institute of
Information Technology
Finland
buntine@hiit.fi

Johan Pouwelse
Dept. of Computer Science
Delft University of Technology
The Netherlands
j.a.pouwelse@ewi.tudelft.nl

ABSTRACT

This paper discusses the ongoing work on *P2P Wikipedia*, a prototype of a personalized tag-based navigation system for Wikipedia content. It is the first open source peer-to-peer (P2P) file sharing system able to deal with large files - like video content - that is being scaled to HTML content as well. The combined techniques in our prototype are the automated calculation of tags from HTML content, an open source personalized P2P file sharing system built on a social network, and the use of incentives for user cooperation to optimize system performance and to function as a training set to the calculation of tags.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

1. INTRODUCTION

Wikipedia is a popular web-based encyclopedia, written and edited collaboratively by volunteers (`wikipedia.org`). In general, the content on Wikipedia pages is text and some small multimedia files - mainly images. The pages do not include extensive video footage. The multimedia version (`commons.wikipedia.org`) doesn't include large video files either. P2P technology presents the possibility for the distribution of large multimedia content. It can reduce the hosting costs and enables the integration of large multimedia files.

There are many methods to search efficiently in text based files with keywords. But this is not that straightforward for video files. Apart from known metadata such as director, title, genre, actors, and year, it is hard to extract keywords from video footage automatically. That is why voluntary tagging is a solution. To illustrate this, consider finding a particular movie, but you have forgotten the title, the names of the actors, or any other metadata that could have helped finding the movie directly. All you remember is that it involved a Citroën DS and a Japanese man. Keyword searching would not lead you directly to *The Goddess of 1967*. But when many users have tagged this movie freely, voluntarily and massively, the chance is much bigger that some have used the tags *Citroën DS* and *Japanese man*, and consequently the chance is also bigger that you will find the movie. Especially the uncommon tags describing specific features in movie scene, e.g. the *Citroën DS* facilitate richer navigation. We believe that tags are an augmentation to keyword searching in video files. They facilitate associative searching and increase the possibility of serendipitous content discovery, especially from the Long Tail [3]. We define a tag as a freely chosen descriptor, or label which refers to one aspect of a movie.

The tag cloud has recently emerged as a popular navigation method through large amounts of tags. The cloud is a representation of the frequency-based relation of tags. An example of a CiteULike tag cloud is shown in Figure 1. Recently other metaphors for visualizing tags have been explored in for instance [6]. The authors have used the metaphors of a waterfall and a river (see Figure 2) to visualize the temporal aspect of tags. This figure shows how the most popular tags of the past one or two years float by. Tags that were used during a longer period float by slower.

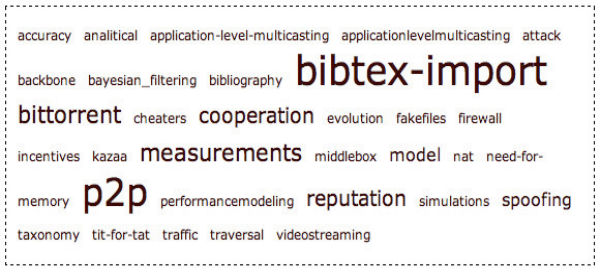


Figure 1: Tag cloud (source: Wikipedia.org)

The two questions we address in this paper are:
1) how to organize and store information in a scalable and efficient manner, and
2) how to stimulate more users to tag more.

As a case study we augment Wikipedia with tags and P2P technology. We believe our approach is generic and can be applied to information organization and storage in general.

OSIR'06 August 11, 2006, Seattle, Washington, USA

Figure 2: Visualizing tags over time (source: Research.yahoo.com/taglines

In this paper we present three contributions. First, an operational open source tool to automatically calculate tags from articles on Wikipedia: Text2Tag, together with the open source tagcloud toolkit taken from `Alvis` [1]. Text2Tag bootstraps the generation of tags and stimulates users to participate in tagging. Second, a fully functional open source P2P file sharing system with a scalable architecture for personalized sharing and real time streaming of large files, like video content: `Tribler` [10]. We are expanding Tribler with tag-based semantic clustering. Third, a user interface with incentives to cooperate, efficient navigation through large amounts of tags, and an exploration of other visualization dimensions that can help make a collection of tags transparent.

2. METADATA, KEYWORDS AND TAGS

Searching and browsing a large collection of video content and not knowing exactly what you are looking for is difficult, especially in a P2P environment, where a problem is the lack of high-quality metadata to enable efficient search. This has been described accurately in [5], stating that the past decade has shown that metadata are often unavailable and searching is best done using the text itself rather than relying on metadata.

Information retrieval from text and audio is relatively easy compared to image and video. Early experiments [17] have shown that distributed information retrieval can be effective through clustering related documents to form the collections. Extracting keywords for images in an embedded environment like a blog or a wikipage can be straightforward, because most of the time the text on the page is the context of the image. But this doesn't hold for large video files, which consist of numerous scenes with each many details, most of which will most likely not be described in contextual text.

Free and voluntary tagging is a good alternative to distributed information retrieval from video files. Moreover, user generated tags can function as a training set for distributed information retrieval from video content. P2P technology has a real potential for video processing. Yet, essential for success is large scale tagging. Websites such as `Flickr.com`, `CiteUlike.org`, and `Del.icio.us` have shown the popularity of tags for search and attracted millions of users. The key is their use of volunteers to augment content with tags. The system performs better if more users participate, because massive and voluntary participation is essential for system success.

However, tags can suffer from ambiguity and arbitrariness. In the absence of a centralized and professional tagging authority, there are no immediate rules for tagging. When everyone who wishes to can contribute to tagging, they will do so in a way that makes sense to themselves and not necessarily to others. The ambiguity of tagging is illustrated by statistics from `Flickr.com`. Table 1 shows the result of our measurement of the various synonyms for the US city of New York.

Table 1: Synomyms of a Flickr tag (Dec'05)

Tag	Number of Photos
nyc	340,000
newyork	228,000
newyorkcity	106,000
new-york	61,000
new-york-city	13,000
ny	67,000
bigapple	2,000

Even though there seem to be initial problems of scalability, and ambiguity of tagging systems, 'with sufficient critical mass, truth would arise from consensus' [16], also known as the *Power of Collective Intelligence*, or the *Wisdom of Crowds* [13]. The advantage is that it can facilitate the task of finding popular tags, and stimulate serendipitous exploration of the tagged universe.

The quality of tags in *P2P Wikipedia* depends on how well they match with the video content. The matching rate is increased when there is mass tagging and when we let people moderate each other in a wiki-style. This is a proven concept: Wikipedia is said to be in the same, or sometimes even higher, league as the Encyclopedia Brittanica [7]. We believe the biggest challenge is stimulating users to tag video content. Our approach to this is exploiting social phenomena, as will be explained in the next section. Furthermore, we ensure a bootstrap for tagging by implementing smart algorithms in our software discussed in Section 3.

2.1 Incentives

In a P2P system it is important for users to voluntarily cooperate. In *P2P Wikipedia* maximal cooperation is essential, because the success of tagging depends on their quantity. Voluntary cooperation does occur in the online world, but we believe we can stimulate more users to cooperate. By using the right incentives we hope to induce users to cooperate and explore the 'tagged universe'. Furthermore, the user interface has to stimulate the active participation in tagging.

Social psychologists identify two basic motivational forces [14] for cooperative behavior. First, *instrumental, or environmentally driven motivation*: people either see the chance of a reward if they cooperate, or fear punishment if they do not cooperate. For instance, websites like `Amazon.com` reward their users with *badges* that show their status and

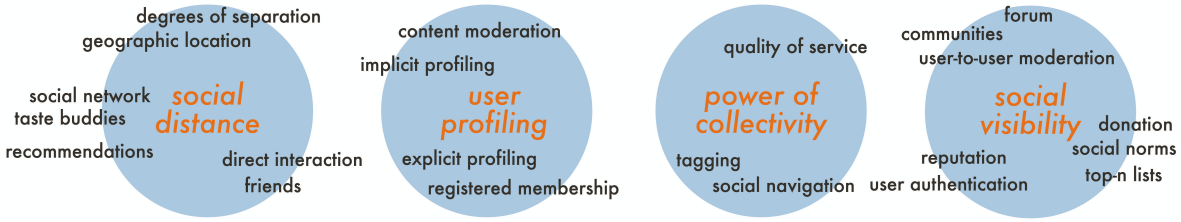


Figure 3: Incentives for cooperation

higher ranks in *top-n lists*. And `eBay.com` lets users moderate transactions. If a buyer or seller has acted contrary to agreement, the other party involved in the transaction can moderate him, thus influencing his reputation. Second, *internally driven motivation*: the influence of personal values in the form of obligation to the group and its rules, and of attitudes relevant to the group, such as commitment, satisfaction, feelings toward group authorities, and loyalty. This is also known as ingroup identification [12], a concept much used in social systems such as `Friendster`, `Orkut`, or `LinkedIn`. How people behave within their group or virtual community is also influenced by their wish to create a positive public self. Factors that influence the way people present themselves positively include the willingness to cooperate, the feeling of belonging, competitiveness, the need to distinguish oneself from others, the possibility to convince others of ones opinion or taste.

In order to understand what incentives are successful in inducing humans to take on roles of cooperation, research has been done in exploring and clustering the features of 22 state-of-the-art applications or websites that rely on the voluntary cooperation of users, among which `Flickr.com`, `Amazon.com`, `IMdB.com`, `Seti@home`, and `Wikipedia.org` itself. The resulting *Taxonomy of cooperation inducing interface features* consists of four clusters of features (see Figure 3) that can be regarded as guidelines for the user interface design of systems depending on voluntary user cooperation. They will be applied to the design of the user interface of *P2P Wikipedia* as well. The clusters - named by the authors - are:

1 Social distance This first cluster explains the influence of the social (and sometimes the geographical) distance in a community of friends and taste buddies on the willingness to cooperate, and how users can profit from social networks. Users can get trusted recommendations from others members that are close to them and interact directly with them. This is a concept used in the MovieLens project (`movielens.umn.edu`).

2 User profiling The features in this cluster are related to an increased willingness to cooperate when the effort they put into the system has a clear advantage, e.g. better recommendations, a higher quality-of-service, or faster access to fresh content. For instance, Amazon users get recommendations based on their past purchases, searching behavior, and explicit rating of items.

3 Power of collectivity This third cluster relates to the before mentioned *Power of Collective Intelligence* and *Wisdom of Crowds*. If more users voluntarily cooperate, it will not only benefit themselves, but the whole community as well. Moreover, it explains that as long as large amounts of users cooperate, none of them have to be true professionals in the expertise in question. Wikipedia is a well known example of the power of collectivity.

4 Social visibility Finally, the fourth cluster shows that users can be very sensitive to how they appear to community members, and to the fact that their actions may influence their reputation within a community (for instance when users can moderate each others behavior). Users will likely try to adhere to the social norms and be more inclined to donate resources to other community members than they would to strangers. For instance, the reputation of eBay users is decisive for others to start a transaction or not.

3. TAGGING THE WIKIPEDIA

The Wikipedia collaborative encyclopedia is our tagging case-study. It is chosen for its availability of content and embedded links, and its large user-base. This section describes relevant aspects of Wikipedia and how we generate tags from the Wikipedia database dump using our `Text2Tag` toolset. It basically is a simple data cleaning technology that normalizes and resolves the within Wikipedia links.

We are well-aware of the difference between user-generated tags and calculated tags, but we choose to call them tags because they will be mixed when the use of *P2P Wikipedia* progresses. The calculated tags we use, nevertheless, have been inserted as links by volunteering authors and editors, thus they are not entirely computer generated. Throughout this paper we will clarify the nature of tags: either user-generated or calculated.

Versions of Wikipedia are available in many different languages. The English language version is the largest with over 930,000 articles in December 2005 with approximately 4.5Gb of uncompressed text (HTML removed) and 580,000 image files including 28,000 with scalable vector graphics. One can perform a targetted search using a Lucene[1] system. Wikipedia also offers topical information on current news daily as well as portals such as the Science Portal. Wikipedia consists of pages with a unique topic name, which can be seen as a unique calculated tag. For example, pages exist for 'democracy', 'coal_mining', and 'Cultural_elements_of_Buddhism'. However, a single Wikipedia page can de-

[1]`http://lucene.apache.org`

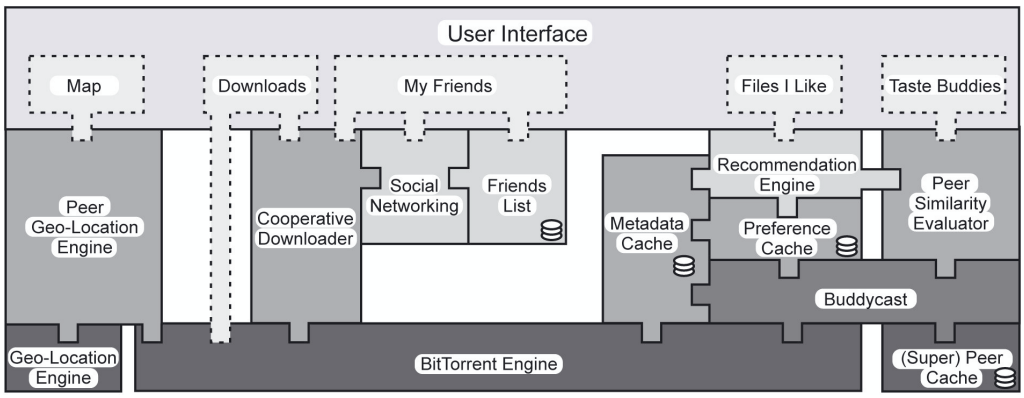


Figure 4: The system architecture of Tribler

scribe numerous subtopics and describe numerous facets of the main topic, and thus cover multiple tags.

Table 2: Ambiguity in Wikipedia pages

Tag	Wikipedia page topic
analytical_engine	Bruce_Sterling
analytical_engine	steampunk
analytical_engine	alternate_history
analytical_engine	The_Difference_Engine
ananda	Cultural_elements_of_Buddhism
ananda	History_of_Buddhism
ananda	List_of_Buddhist_topics

Table 2 shows some examples of the relation between tags and Wikipedia page topics. In December 2005 Wikipedia included roughly 32,000,000 links between pages, and 790,000 redirects from variant topic names (e.g., 'Abel' to 'Cain_and_Abel'). Moreover, the link text associated with tags, the text an author has written for a link, is every bit as varied as the tags in Flickr or other systems. Thus our association of the link to a tag means we have, in effect, had the synonym problem solved for us.

We developed software to generate tags from Wikipedia as a bootstrap for user-generated tags. These tags are the title text for pages that are not disambiguation pages or stubs, that have more than 4 in-links (where 4 is arbitrarily chosen), and that may also be category pages. A page containing a link (possibly through a redirection) to such a page is said to contain the 'tag'.

The challenge is not only generating tags, but also organizing them into top-tags, sub-tags, subsub-tags, and adding weights. We implemented the generic `GenerateTopTags` function to generate tags. This function can generate both top-tags, sub-tags, and subsub-tags. It increases freedom in searching and exploring content, and this bootstrap should stimulate more users to generate more tags for their own content. Candidate subtags and matching documents are generated using both an inverted and a forward index (i.e., bags of tags for each document). Top-tags are ranked using a PageRank score [8] that downgrades the time tags. Time is richly tagged in Wikipedia, so tends to dominate other concepts. Ranking a candidate list of sub-tags is done by combining a standard `idf` score [4] relative to the tag. `Idf` is inverse document frequency, and the sub-tags frequency is taken from the set of documents having the major tag, not the full collection. Ranking a candidate list of documents matching a particular query made up of tags is done again with a standard `tf*idf` score.

4. NEXT-GENERATION P2P

We are currently working on generalizing P2P file sharing to supporting content distribution in general. For this we implemented an operational open source P2P system called *Tribler* [10]. Since its release in March 2006, there have been over 60,000 downloads (`sf.net/projects/tribler`).

Tribler is based on the popular Bittorrent protocol. Source code is available from `Tribler.org`. Real-time MPEG4 streaming is merged with Bittorrent [11]. Furthermore, we are merging our file sharing system with web technology, thus creating a decentralized Wikipedia. Several possible business models exist, the donation-based approach exemplified by Wikipedia donation rallies, a model where targetted advertisements pay for the hosting costs, and variations to these themes, e.g. `Amazon.com`. We show another model where users tag video content, moderate existing content, and provide the resources for persistent storage, publication, and distribution. This model is implemented in Tribler, which will be explained in the next section.

4.1 Tribler architecture

In this section we present the architecture of our Tribler social-based P2P file-sharing system, which is built on top of the Bittorrent protocol. Figure 4 depicts the architecture of the Tribler network client. Rectangles represent client modules. The extrusions represent *make-use-of* relationships. To achieve backward compatibility with the existing Bittorrent network, while offering our users extended functionality, we only made modifications and extensions to the Bittorrent client software. Our system is based on the ABC open-source client [2]. By extending this popular client we aim to have a large user base in a relatively short time, besides having a tested code base for our implementation.

Social phenomena The prime social phenomenon we exploit in Tribler is an analogy to evolutionary biology: kinship

fosters cooperation [9]. This phenomenon is part of the *social distance* and *social visibility* clusters as explained in Section 2.1. Kinship is interpreted as friendship or belonging to a community, because genetical relations are not taken into account. Similar taste for content can be one of the foundations for an online community with cooperative behavior, instead of remaining an ad-hoc group of non-cooperating strangers.

It is beneficial to have friends in Tribler, because users can donate their idle bandwidth to friends, thus boosting that friends' downloadspeed in a cooperative download, as can be seen in Figure 5.

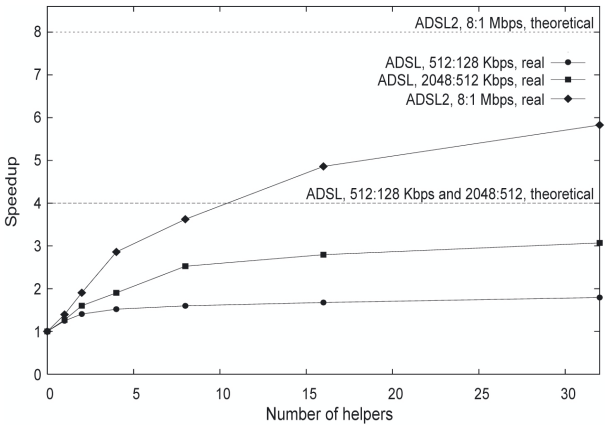


Figure 5: Downloadbooster in Tribler

The *Social Networking* module in Figure 4 is responsible for storing and providing information regarding social groups (the group members, their recently used IP numbers, etc.). This will be discussed in the next paragraph.

Tag-buddy based content discovery Locating content is critical for P2P systems. Current solutions are based on one or a combination of query flooding, distributed hash-tables, and semantic clustering. We take a next step by connecting *people* with similar tastes instead of focusing on *files*, and by using full metadata replication. In Tribler we exploit the fact that people with similar tagging behaviour - also known as *tag buddies* - have related taste.

Using the *Files I Like* module, each peer indicates its preference for certain files and their associated tags. By default, the preference list of a peer is filled with its most recent downloads. We have developed an algorithm called *Buddycast* that employs an epidemic protocol [15] to exchange preference lists using the overlay swarm and that can efficiently discover a user's tag buddies. The *Peer Similarity Evaluator* module in Figure 4 is able to compare similar preference lists.

4.2 P2P Wikipedia

Our software will enable the extension of Wikipedia with multimedia and tags. We are using our `Text2Tag` toolset to import Wikipedia tags into Tribler. Due to the excessive Wikipedia bandwidth usage it has not been possible to augment pages with extensive video footage. The required servers and Internet connection can not be supported by the Wikipedia donation-based approach only. But by integrating the proven Bittorrent technology we can reduce bandwidth bottlenecks and create a scalable system.

There are four key extensions needed on Tribler for *P2P Wikipedia*. First, the ability to display Wikipedia content inside Tribler, thus add an embedded web browser. Second, remove the Bittorrent tracker from the content discovery architecture using epidemic protocols [15]. Third, eliminate the need for .torrent files by using Merkle hashes and embedding these into a URL. Fourth, build a version management system on top of the Bittorrent content layer to enable collaborative editing.

5. PROTOTYPE

In this section we explain the user interface of the tag-based navigation prototype for *P2P Wikipedia*. Tag-based navigation can be an additional way to explore Wikipedia, especially for video files. *P2P Wikipedia* can offer users personalized recommendations based on their taste. A user's taste is learned from creating new content, tagging, moderating, searching, and browsing. The *Buddycast* algorithm then calculates the user's taste buddies - also known as *tag buddies* in *P2P Wikipedia* - to create a sense of belonging to a community. From all this it will also be easier to calculate recommendations, as is done in the *Tribler* system, and tag-to-tag similarity. They both result in a much richer and more serendipitous exploration of the 'tagged universe'.

The main navigation screen in Figure 6 shows the calculated tag cloud with 50 - 100 top Wikipedia tags on the left, arranged alphabetically. Their relative ranking is expressed by font-size. Navigation in this prototype is keyword searching, and tag-browsing. The area on the right is used for personalized settings, containing:

a) My Tags. Summarizing the user's most often used tags. Like the tag cloud on the left, this personal tag cloud is arranged alphabetically, and the ranking is expressed in font-size.

b) Recent Tags. A list of 12 most recently viewed tags (xn-1...n-12). The user can perform an AND operation of one of these tags with the currently viewed tag cloud by clicking the button behind that tag.

c) Friends and Tag Buddies. An overview of friends, friends-of-a-friend and buddies that are currently online. Information about a user's friend or friend-of-a-friend comes from the integration of an existing social network (not yet built in the prototype). Showing this social network will de-anonymize the system and stimulate contribution. An example of *social visibility* is that users are rewarded with stars for their cooperation.

Figure 7 shows the tag cloud resulting from the search operation 'holland AND tourism' and the directly matching Wikipedia results. Floating the mouse over a tag brings about two things (see Figure 7). First of all, directly related tags in the tag cloud are highlighted. The tag *amsterdam* has a number of directly co-occuring tags, e.g. *schiphol* and

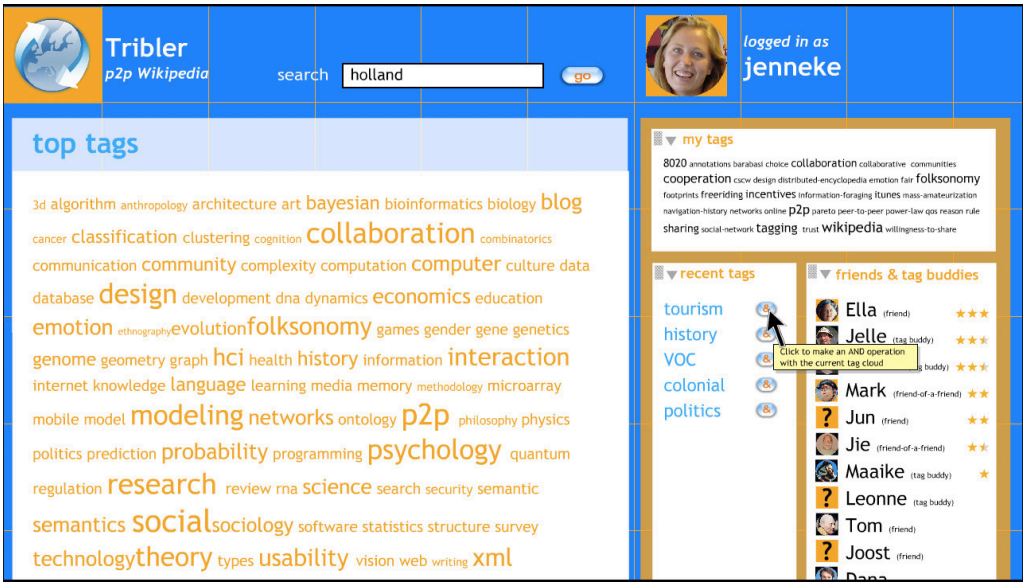


Figure 6: Main navigation screen

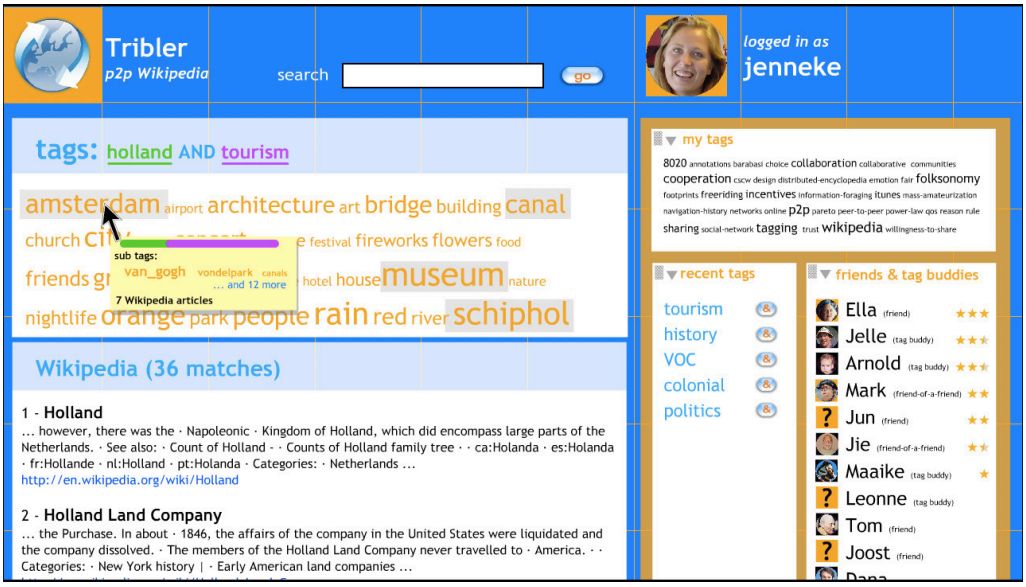


Figure 7: Mouse-over showing sublevel

rain. Secondly, information about the tag is shown in a tooltip: The tag *amsterdam* in this figure has a number of sub-tags and Wikipedia articles. The colored bar in the tooltip shows how much the tag relates to the two keywords *holland* and *tourism* relatively. Both the highlighting and the tooltip facilitate more efficient navigation, because information is available in advance.

6. CONCLUSIONS AND FUTURE WORK

We have created a way of searching within video content that can be added to Wikipedia. In this paper we explained why, when, and how tag-based navigation can augment traditional keyword searching for `Wikipedia.org`, and why cooperation is needed from the users to do so.

We have also described why P2P technology is ideal for the distribution of video content. P2P technology accumulates distributed resources and can also be applied to HTML content. We presented our vision for the design of the user interface which incorporates incentives to cooperate.

In the progressing development of our code for tag-based navigation, two aspects remain important. First of all, we will have to work out ways to get information retrieval working in a P2P manner. P2P search is difficult, but coherence is found in the Tribler framework. And secondly, our tools for tag-based navigation have proven to be very powerful and we will be setting them up for open source information retrieval.

Tribler is available for download from `Tribler.org`. We are planning controlled experiments in a natural environment with the user interface on a few dozen users in the second half of 2006. This will enable us to test and improve the user interface, and the efficiency of the incentives to cooperate and tag-based navigation. Furthermore, our ambition is to merge and integrate all information from `Wikipedia.org`, `Del.icio.us`, and `CiteUlike.org` into a single coherent P2P system with tags as the organizing principle.

6.1 Acknowledgments

The authors would like to thank Pekka Uronen and Jaakko Lofstrom from the Helsinki Institute for Information Technology for their help and input on the dataset and the demo of *P2P Wikipedia*.

7. ADDITIONAL AUTHORS

Huib de Ridder (Delft University of Technology, `h.deridder@tudelft.nl`) and
Piet Westendorp (Delft University of Technology and Eindhoven University of Technology, `p.h.westendorp@tudelft.nl`).

8. REFERENCES

[1] Website at `http://www.alvis.info`.

[2] ABC client at `http://sf.net/projects/pingpong-abc`.

[3] C. Anderson. The long tail. *Wired Magazine*, October 2004.

[4] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[5] D. Bulterman. Is it time for a moratorium on metadata? *IEEE MultiMedia*, 11(4):10–17, 2004.

[6] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing tags over time. In *15th International World Wide Web Conference*, May 2006.

[7] J. Giles. Internet encyclopaedias go head to head. *Nature*, December 2005.

[8] A. Langville and C. Meyer. Deeper inside PageRank. *Internet Mathematics*, 1(3):335–400, 2004.

[9] E. Pennisi. How did cooperative behavior evolve? *Science*, 309(5731):93, July 2005.

[10] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. van Steen, and H. Sips. Tribler: A social-based peer-to-peer system. In *5th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, February 2006.

[11] J. Pouwelse, J. Taal, R. Lagendijk, D. Epema, and H. Sips. Real-time video delivery using peer-to-peer bartering networks and multiple description coding. In *IEEE Conference on Systems, Man & Cybernetics*, October 2004.

[12] S. Reid and M. Hogg. Uncertainty reduction, self-enhancement, and ingroup identification. *Personality and Social Psychology Bulletin*, 31(6):804–817, June 2005.

[13] J. Surowiecki. *The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations*. Anchor Books, 2005.

[14] T. R. Tyler and S. L. Blader. *The Antecedents of Cooperative Group Behavior*. Psychology Press, 2000.

[15] J. Wang, J. Pouwelse, J. Fokker, and M. Reinders. Personalization of a peer-to-peer television system. In *4th European Interactive TV Conference (EuroITV)*. EuroITV, May 2006.

[16] A. Weiss. The power of collective intelligence. *netWorker*, 9(3):16–23, 2005.

[17] J. Xu and W. Croft. Cluster-based language models for distributed retrieval. In *Research and Development in Information Retrieval*, pages 254–261, 1999.

Web Recommender System Implementations in Multiple Flavors: Fast and (Care-)Free for All

Olfa Nasraoui, Zhiyong Zhang, and Esin Saka
Department of Computer Engineering and Computer Science
Speed School of Engineering
University of Louisville
Louisville, KY 40292

ABSTRACT

In this paper, we present a systematic framework for a fast and easy implementation and deployment of a recommendation system for one or several Websites, based on any available combination of open source tools that include crawling, indexing, and searching capabilities. The supported recommendation strategies include several popular flavors such as content based filtering (straight forward), collaborative filtering (more complex), rule-based, as well as approaches that deal with meta-content, (non-textual) attributes and ontologies, and other variants that include meta-attributes about the user, such as elaborate user profiles, as well as business strategy rules. The biggest advantage of this approach is that for content-based filtering, it allows client or proxy controlled integration of several websites.

1. INTRODUCTION AND MOTIVATIONS:

Recommender systems have been attracting more and more attention as a suitable approach to counteract information overload and help the users of the Web information space find what they need faster. While there have been significant advances in the theoretical and algorithmic aspects of recommender systems, there are no clear and systematic implementation approaches that have been published, as most of these tend to be cloaked under thick walls of corporate secrecy. Also, typical commercial packages may come at a hefty price that may not be affordable to smaller or start-up e-commerce and e-media websites, including the increasing number of purely non-profit community information Websites and forums that cannot even be classified yet as businesses.

In this paper, we present a systematic framework for a fast and easy implementation and deployment of a recommendation system on a Website (or several affiliated or subject-specific websites), based on any available combination of open source tools that include crawling, indexing, and searching capabilities. The supported recommendation strategies include several popular flavors such as content based filtering (straight forward), collaborative filtering (more complex), rule-based, as well as approaches that deal with meta-content (non-textual) attributes and ontologies, and other variants that include meta-attributes about the user, such as user profiles, as well as other business strategy rules. Note that this paper is mainly about ways to *easily "implement"* (*existing*) recommendation strategies by using a search engine software when it is available, which therefore can be expected to benefit research and real life applications by taking advantage of search engines' *scalable* and *built-in* indexing and query matching features, instead of implementing a strategy from scratch.

1.1 Advantages of the Search Engine Tweaking Approach to Recommender System Design

- **Multi-Website Integration by Dynamic Linking:** Easily and seamlessly integrate multiple websites, even those that are hosted on disparate web servers. This facilitate the *dynamic, personalized, and automated linking* of partnering or affiliate websites (e.g. several educational, research, and outreach websites related to a common topic).
- **Giving Control Back to the User or Community instead of the wesbite/business:** The content-based filtering approach (and to some extent the collaborative filtering variations when implemented on a proxy) are no longer confined to the realm of the server, and thus no longer have to serve only the interests of a particular website or business. This means that the user (or a community of users) can install such a system on their own computers (or a proxy machine in case of a community) to unify and dynamically link any group of websites that are relevant to their interests. A group of users (for instance students in the same school district, workers in the same company, or college students in the same university department) can similarly *share* their usage sessions to enrich the collaborative part of the recommendations, if they install the proposed system on a common proxy.
- **The Open Source Edge:** There are currently several available open source implementations for search engines which can form the foundation for several types of recommendation strategies as will be explained in this paper. Being open source, they offer the potential for improvements by the open source community, as well as providing a good, easy, and open platform for teaching and research, with potential to be extended and enhanced by different teams working on Web personalization and Web search.
- **The Information Retrieval Legacy:** Being based on information retrieval, this approach allows inheriting from all previous research and advances in this area to also benefit the field of recommendation systems. This includes inheriting from advances in such areas as: Information Filtering, Personalized Information retrieval, and Query reformulation and expansion.
- **The Semantic Web Vision:** In the same way that different advances have taken place in the field of Information retrieval, significant efforts and developments are already taking place in the area of the *Semantic Web*, where the Web pages' content is handled based on their *meaning, and not just their lexical content.* Developments that can improve the quality of information retrieval by taking semantics into account, can now also have an impact on recommender systems in a seamless manner. For example, *thesauri* will be able to get integrated easily. Thus, the power of Semantic Recommender Systems can be expected to benefit from and evolve with the Semantic Web.

The rest of this paper is organized as follows. In Section 2, we give an overview of Web personalization including recommender systems. In Section 3, we give an overview of the structure and functionalities of a typical search engine implementation. In Section 4, we explain the procedure needed to harness a search engine to implement several types of recommender systems. In Section 5, we present some implementations of the proposed methodologies, and finally make our conclusions in Section 6.

2. BACKGROUND ON WEB PERSONALIZATION

In the late 90s, Jeff Bezos, CEO of Amazon once said, If I have 3 million customers on the Web, I should have 3 million stores on the Web [21]. Web personalization tailors a user's interaction with the Web information space based on information gathered about them. The recommendation process follows a decision making process that typically ends up with generating dynamic Web content on the fly, such as adding hyperlinks to the last web page requested by the user, in order to facilitate access to the needed information on a large website [21] [13] [16]. It is usually implemented on the Web server, and relies on data that reflects the users interest implicitly (browsing history as recorded in Web server logs) or explicitly (user profile as entered through a registration form or questionnaire). The *implicit* approach is the focus of the work presented in this paper. Personalization can be used to achieve several goals, ranging from increasing customer retention and loyalty on e-commerce sites [21] to enabling better search by making results of Web information retrieval/search more aware of the context and user interests [10]. In addition, personalization can help convert browsers into buyers, increase cross-sell by recommending items related to the ones being considered, improve web site design and usability, help visitors to quickly find relevant information on a large website, and more generally help businesses maintain a more effective Customer Relationship Management (CRM) [7]. Recommender systems have already found some success in real e-commerce applications such as Amazon [12] and CDNow [1], where they are used to recommend to online shoppers, products and services that they might otherwise never discover on their own. There have also been several pioneering research system prototypes, such as Syskill and Webert [18], PHOAKS [24], Fab [3], and GroupLens [11] [20]. Recommender systems can be classified in a variety of ways [21] [17] [4] depending on their inputs and on the recommendation algorithm used, including:

- Content-based or Item-based filtering: Content-based filtering systems recommend items to a given user, which are deemed to be similar to the items that the same user liked in the past. Item similarity is typically based on domain specific item attributes (such as author and subject for book items, artist and genre for music items). Classical examples include Syskill and Webert [18], and Fab [3].

- Collaborative filtering: Based on the assumption that users with similar past behaviors (rating, browsing, or purchase history) have similar interests, a collaborative filtering system recommends items that are liked by other users with similar interests [21]. This approach relies on a historic record of all user interests such as can be inferred from their ratings of the items on a website (products or web pages). Rating can be explicit (explicit ratings, previous purchases, customer satisfaction questionnaires) or implicit (browsing activity on a website or clickstreams). Typical examples include GroupLens [11][20]. Computing recommendations can be based on lazy or eager learning to model the user interests. In lazy learning all previous user activities are simply stored, until recommendation time, when a new user is compared against all previous users to identify those who are similar, and in turn generate recommended items that are part of these similar users interests. On the other hand, eager learning relies on data mining techniques to learn a summarized model of user interests (a decision tree, clusters/profiles, etc) that typically requires only a small fraction of the memory needed in lazy approaches. One particular eager modeling approach, that has lately received increasing attention, is known as *Web Usage Mining* [15][23][22]. *Web Usage Mining* consists of applying machine learning or data mining techniques to discover interesting usage patterns and statistical correlations between web pages and user groups. This learning frequently results in automatic user profiling, and is typically applied offline, so that it does not add a burden on the web server. Collaborative filtering methods are completely independent of the intrinsic properties of the items being rated or recommended. In particular, items that may be hard to describe using attributes, such as video, audio and images, as well as semantically rich text data can still be recommended based on latent similarities that are only captured through the *social* process of collaborative filtering, hence, often suggesting completely *new* types of items that are *different* from, and yet *associated* with previously rated items.

- Knowledge Engineering or Rule-based filtering: In this approach, used frequently to customize products on e-commerce sites such as Dell on Line, the user answers several questions, until receiving a customized result such as a list of products.

- Demographic recommender systems: In this approach, items are recommended to users based on their demographic attributes. The recommendations can be based on handcrafted stereotypes derived from marketing research or on machine learning techniques [17] that learn to predict users' preferences from their demographic attributes.

- Hybrids: Each recommendation strategy has its own strengths and weaknesses. Hence, combining several recommendation strategies can be expected to provide better results than either strategy alone [17][4]. Most hybrids work by combining several input data sources or several recommendation strategies.

3. OPEN SOURCE SEARCH ENGINES - THE CASE OF "NUTCH"

Before we present our systematic procedure to make a search engine deliver recommendations, we will briefly explain the architecture of Web search engines. Most search engines rely on four crucial components that perform the following functions usually in this order: ***(i)*** **Crawling:** A crawler retrieves the web pages that are to be included in a searchable collection, ***(ii)*** **Parsing:** The crawled documents are parsed to extract the terms that they contain, ***(iii)*** **Indexing:** A *reverse index* is typically built that maps each parsed term to a set of pages where the term is contained, ***(iv)*** **Query matching:** After the index has been built, input queries in the form of a set of terms can be submitted to a search engine interface or to a query matching module that compares this query against the existing index, to produce a ranked list of results or web pages. Below, we focus on two open source products that enable a fast and free implementation of Web search, first a text search engine library called *Lucene*, and then the Web search engine, *Nutch*, that is built on the Lucene text search library.

3.1 Lucene

3.1.1 General overview and special features

Apache Lucene [6] is an open source project of the Apache

Software Foundation.[1] *Lucene*[2] is a high-performance, full-featured *text search engine library* written in Java, and can support any application that requires full-text search, especially cross-platform. Examples of using Lucene include *Inktomi* and *Wikipedia*'s search feature. Lucene offers powerful features through a simple API, including scalable, high-performance indexing, as well as powerful, accurate and efficient search algorithms. It is also available as Open Source software under the Apache License which allows using Lucene in both commercial and Open Source programs. Of particular interest to this paper are the flexible and powerful features of Lucene's search algorithms[3] : ranked searching; powerful query types: phrase, wildcard, proximity, fuzzy, range, and more; fielded searching (e.g., title, author, contents), date-range searching, sorting by any field, multiple-index searching with merged results, and allowing simultaneous update and searching.

3.1.2 Query Syntax

A query in Lucene is broken up into *terms* and *operators*, with two types of terms: *Single Terms* and *Phrases*. A *Single Term* is a single word such as "test" or "hello". A *Phrase* is a group of words surrounded by double quotes such as "hello dolly". Multiple terms can be combined together with *Boolean* operators to form a more complex query. Lucene also supports *fielded data*. A search can thus be performed by specifying a field, or by using the default field. The field names and default field are implementation specific. Lucene queries can also *boost* a particular term so that it weighs more heavily in the matching process, for example the query "jakarta^4 apache" will cause documents containing the term "jakarta" to be considered more relevant compared to the ones containing the term "apache". Note that the boosting and (default) Boolean OR based queries can easily be used to form *vector space queries* (the term frequencies become the boosting weights)[4] or by sorting the results of a Boolean OR query based on vector space similarity, as described for example in the TREC & HARD' 05 record in [8][9].

3.1.3 Scoring

Given an input query, Lucene scores the results based on the TF-IDF measure where the documents are represented in the vector space model.

3.2 Nutch

Nutch (http://lucene.apache.org/nutch/) is an open source *Web search software*, built on Lucene Java, that adds web-specifics, such as a *crawler*, a *link-graph database*, *parsers for HTML* and *other* document formats (such as pdf, Microsoft Powerpoint and Word, plain text, etc). Other document types can be parsed by programming their specific *plugin*. Nutch maintains a Database of pages and links. For each web page that is indexed, only the *URL* and *title* are stored, while the *anchor* and the *content* are only indexed. A *document* is a sequence of *Fields*. A *field* is a ¡*name, value*¿ pair, where *Name* is the name of the field, e.g., title, body, subject, date, etc; and *Value* is text. Field values may be stored, indexed, analyzed (to convert to tokens), or vectored. Lucene's index is an *Inverted Index* that maps a term to a *field ID*, and a set of document IDs, with the position within each document. Nutch uses Lucene search which supports both *primitive queries* (term, phrase, and Boolean) and *derived queries* (prefix and/or wildcard). Given a query, Nutch by default searches *URLs, anchors, and content* of documents, and also rewards for *proximity* of the query words in the documents. Finally, we note that Nutch also uses *N-Grams* to index very common terms with their neighbors, which in turn improves *phrase search*.

[1]http://www/apache.org
[2]http://lucene.apache.org/
[3]http://lucene.apache.org/java/docs/features.html
[4]http://www.cs.virginia.edu/~xj3a/research/TREC/index.htm

4. PROPOSED METHODOLOGY

4.1 Requirements for tweaking a search engine to work like a recommender system

There are two principal requirements to being able to use a search engine to provide recommendations:

- **An index:** The source of the recommendations must be stored or rather indexed in a format that is easy to search.
- **A querying mechanism:** Regardless of the recommendation strategy and regardless of what has been indexed, the *input* to the recommendation procedure *must be transformable into a query* that is expressed in terms of the entities upon which the index is based. For instance if the index maps every keyword in a given vocabulary to a set of pages that contain this keyword, then the query needs to be expressed in terms of these keywords.

Hence, there appear to be two components that can define how the search engine will deliver recommendations: the *index* and the *query*. In order to realize a given recommendation strategy, it is therefore necessary to at least define how each of these components will be handled. In the following subsections, we assume that both a raw session and a simple collaborative profile consist of a *set* of indexed items/Web pages P_i (note that this set can also be considered as a *vector* in the space defined by all possible indexed items), thus

$$\textbf{raw-session} = \{P_1, P_2, ..., \text{etc}\}$$

$$\textbf{profile} = \{P'_1, P'_2, ..., \text{etc}\}$$

4.2 Methodologies for tweaking a search engine to work like a recommender system

4.2.1 Content-based filtering

The content-based filtering approach is straightforward in case the recommended items are HTML or text-based pages. The idea behind content-based filtering is that given a few pages that a user has viewed, the system recommends other pages with content that is similar to the content of the viewed pages. Most content-based filtering systems store an item-to-attribute matrix that describes every item as a feature vector with several attributes. In the case of textual content, the bag of words is a simple and popular approach to representing the items (in this case web pages). Given a set of items that have been viewed or rated by a user, the recommendation system will then compute the similarity between the rated items and all other items to determine which items are most similar to what the user showed interest in, and then recommends the top items. However, a search engine works by reverse-indexing the items (in this case, web pages), and then comparing a submitted query to the indexed items in order to determine the closest items. For our purpose, the recommendation process amounts to performing the following two steps:

1. **Preliminary Crawling and Indexing (*done offline*):** The first step will consist of crawling the website(s) that will contribute to the *content* of the recommendations, and then forming a reverse index that maps each keyword to a set of pages in which it is contained. Since we plan to compare a query to a set of web pages in *vector space* format, we store the most frequent terms in each document as a *vector field*, that is indexed and used later in retrieval by submitting a fielded query.
2. **Query Formation and Scoring:** This step essentially consists of transforming a new user session into a query that can be understood by the search engine. First, each URL in the user session is mapped to a set of content terms that are most characteristic of this URL (top k frequent terms) using an added package `net.nutch.searcher.page`

Then these terms are combined with their frequencies to form a query vector, that gets submitted to the Nutch search interface as a *Fielded* query, i.e. the query vector is compared to the indexed Web document vector field. Finally, the results are ranked according to the cosine similarity of their content with the query vector in the vector space domain (introduced as a modification of the default scoring mechanism of `SortComparatorSource` in the `LuceneQueryOptimizer` class which is part of the package `net.nutch.searcher`). We are currently also in the process of experimenting with a Boosted disjunctive Boolean query[5] as discussed within the context of information retrieval in [8][9].

session → URLS → terms → fielded query vector → results (ranked according to cosine similarity (result vector, query vector))

In the *final query formation*, in order to give preference to the most recent context (i.e. web page requests), the content vectors of the pages in the session are weighted differently, so that more recent pages receive higher weights than older pages. This has the effect of *forgetting* the content of older pages in the session compared to the more recent requests. This process is shown in Fig. 1.

Once the query has been formed, it is delivered to the search engine for further processing, and the search engine returns a ranked set of items/web pages that are sorted by the *cosine similarity* between the *vector of TF-IDF-weighted* terms of the query (which in turn represents a user's session) and the *TF-IDF weighted* terms of the indexed pages. Hence the results accomplish the goal of content-based filtering. The user session is transformed into a query that accumulates all the term weights of the pages that are visited. However, the weight w_1 of an older page P_1 gets weakened with the arrival of each new page in the session by multiplying it by a forgetting factor F in $(0, 1)$ as follows: $w_1 \leftarrow F.w_1$. The *cumulative session to content query transformation* is illustrated in Fig. 2. Note that some essential features of text retrieval, used as part of *Nutch*, such as *TF-IDF* are inherited in the proposed content-based filtering method. This is desirable as we do not want the very popular terms in a Web collection to dominate the matching process. For example, a website or a set of affiliated websites that are devoted to the the topic "Solar Astronomy" are expected to contain a large number of Web pages containing either one of these terms. Thus, these terms may not be useful in distinguishing between the different pages. The TF-IDF measure will suppress the contribution of such common or popular terms exactly as desired, and focus instead on other terms that help better distinguish between the pages.

4.2.1.1 Anchor and URL matching: .

An option supported by most search engines would be to include the anchor information of the viewed web pages (the anchor text is the text used in hyperlinks that point to a given page) when formulating the queries. Here, once the anchor data is added to the query vector (in the same way as the actual content terms), they are submitted as an (anchor+content) query, and this query may be matched against the content index or the content and anchor index. Most search engines end up comparing a given query not only to the content of the indexed Web pages, but also to their anchors, and even their URLs. For instance, given a query, Nutch by default searches *URLs, anchors, and content* of documents, and also rewards for *proximity* of the query words within the documents. We consider this to be a powerful feature for Content-based filtering recommendations, since the item matching is not only based on the intrinsic content, and takes into account proximity of the terms, but it is actually extended to the "location label" or URL of the item and its position in a website hierarchy and even rewards similarly named items in different areas of the website. Anchors can also play an important role in *enriching* and to some extent, *disambiguating* intrinsic content that is *lexically* similar. We see this as an added bonus of the search engine based realization of a recommendation system, since the anchor and URL indexing and matching features are a natural component that is taken for granted in many search engines, and yet could be considered a hassle to implement from scratch as part of a recommendation approach.

[5]http://www.cs.virginia.edu/~xj3a/research/TREC/index.htm

Figure 1: Cumulative *session to content query transformation* in the Content-based Filtering approach

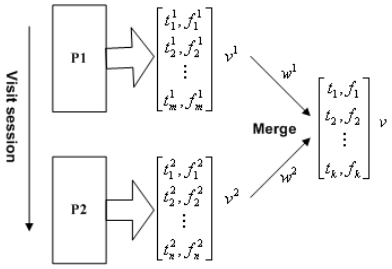
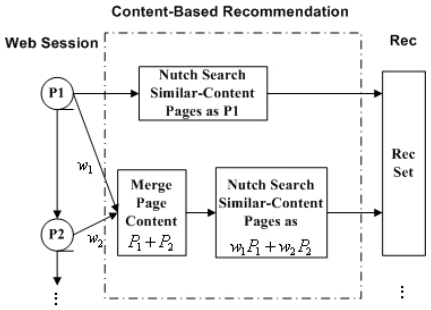


Figure 2: Content-based Filtering implemented via a Search Engine



4.2.2 Collaborative filtering

Compared to content-based filtering, the collaborative filtering approach is more tricky since it involves multiple users and forming neighborhoods of these users, however there are two ways that a crawl-index-search engine can be tweaked to deliver collaborative filtering recommendations:

4.2.2.1 Full-Fledged Collaborative Filtering.

A possibly complex approach would involve first creating an index consisting of the user sessions (instead of web pages) in the URL domain (instead of the keyword domain), then using this index as the basis for query matching (again in the URL domain), thus submitting a query in the form of a set of URL IDs instead of a set of words, and retrieving as result, a set of similar sessions. Finally, the frequency of occurence of the pages (URLs) contained in all the retrieved similar sessions are accumulated, and the pages/URLs are sorted in decreasing order of their relevance. This forms the final result. This final operation needs to be programmed as a plugin to the query matching component. Note that the query transformation component of this search engine tweak is straightforward, i.e.:

session = query

Also, this methodology amounts to pure collaborative filtering, where the content of the pages plays no role in the recommendations, and *only* the similarities between the user *activities* drive the recommendations. From a search engine tweaking point of view, it is important to note that this approach puts its entire leverage on *the indexing component* and not the query formation, since the sessions are indexed as if they were documents, and hence the crawling component becomes obsolete. The only difference compared to content-based filtering is that the *index* is in the *URL domain* instead of the keyword domain. That is, that the query matching relies on a vector space representation of the sessions in the URL domain instead of a vector space representation of the documents in the keyword domain.

As a legacy from the TF-IDF scoring approach used in scoring, our collaborative filtering approach will inherit its suppression of very popular URLs that are visited frequently in most sessions, indeed a very desirable side-effect. This is because most popular pages tend to include navigational pages that tend to be located at the crossroads of many paths through the website (and which tend to be excessively linked to on the website).

4.2.2.2 *Hybrid Content via Collaborative Filtering with Cascaded/Feature* Augmentation *Combination.*

Hybrid Content via Collaborative Filtering amounts to performing the following steps:

1. **Preliminary Crawling and Indexing (*done offline*):** This preliminary step will consist of crawling the website(s) that will contribute to the *content* of the recommendations as in content-based filtering, and then forming a reverse index that maps each keyword to a set of pages in which it is contained. This step is identical to Step 1 in 4.2.1. This index is called *Index 1* (in term space).

2. **Preliminary Indexing of Prior User Sessions using Method described in Full-Fledged Collaborative Filtering in Sec.** 4.2.2.1 **(*done offline*):** Create an index consisting of the user sessions (instead of web pages) in the URL domain (instead of the keyword domain), this will allow matching a new user session in the URL domain to similar user sessions in the URL domain. We call this index *Index 2* (in URL space).

3. **Matching New User Session to Previous Sessions to Form a Collaborative Session:** The next step is to match a new user session to the existing sessions in *Index 2* in order to retrieve the top N similar sessions. The frequency of occurence of the pages (URLs) contained in all the retrieved similar sessions are accumulated, and the pages/URLs are sorted in decreasing order of their relevance. This forms an *intermediate* result that is identical to the final result of *Full-Fledged Collaborative Filtering*. We call this result a collaborative session, because it has the same structure as the original raw session, and will further go through content enrichment in the next step.

4. **Query Formation and Scoring:** The final step proceeds similarly to Content-based filtering as explained above. That is, first we transform this collaborative-session into a content query by merging the Web pages' content as in content-based filtering, and then submit this content query to the search engine that matches it against its content index, *Index 1*.

The entire transformation process can be summarized as follows:

session (submit to *Index 2* in URL space) → similar sessions → collaborative-session → URLS → terms → fielded query vector (to submit to *Index 1* in term space) → results (ranked according to cosine similarity (result vector, query vector))

Note that this approach will result in a *hybrid* (*collaborative and content*) recommender system, since the final recommendations will also take content into consideration. The *collaborative* ingredient comes from the fact that the content is not only formed from the current session, but also combines the content from a neighborhood of *"similar"* sessions. Furthermore, this approach may be classified into the family of *cascaded hybrid recommendation strategies* according to [4], as well as *feature augmentation hybrid recommendations,* also discussed in [4]. The latter was defined as "*methods where the output from one technique is used as an input feature to another*", which is what is performed in this case. At the same time, *cascaded* hybrids were defined in [4] as "*One recommender refines the recommendations given by another*", which is also the case of the described approach because *both* stages result in a set of URLs (thus recommendations).

We call this approach *content-via-collaborative-filtering*, and it is illustrated in Fig. 3. Compared to *pure Content-based Filtering*, this approach *first modifies the session* by including items (pages) of interest to *similar* users, and then performs content-based filtering on the resulting session. This approach is reminiscent of Pazzani's *Collaborative-via-Content* recommendation approach [17], however the latter performs the two steps in *reverse* order, i.e. Content-based filtering, followed by Collaborative filtering. When it comes to the *final query formation*, the only difference between the combined approach and the pure Content-based Filtering approach is that there is *no forgetting* of the content of older pages in the session compared to the more recent requests, though this feature can be accomplished easily in Step 3 if *older* pages are assigned lower weights.

4.2.2.3 *Hybrid Content via Profile-based Collaborative Filtering with Cascaded/Feature Augmentation Combination.*

Hybrid Content via Profile-based Collaborative Filtering amounts to performing the following four steps:

1. **Preliminary Mining of Usage Profiles (*done offline or online with scalable Web usage mining techniques*):** A simpler profile based approach assumes that we have already mined collaborative user profiles [5, **?**, **?**], or possibly that we have been mining usage micro-clusters incrementally, for example in a streaming framework [14]. Once a set of collaborative user profiles is available, they will form the basis for collaborative filtering.

2. **Preliminary Crawling and Indexing (*done offline*):** Another preliminary step will consist of crawling the website(s) that will contribute to the *content* of the recommendations. This step is identical to Step 1 in Sec. 4.2.1.

3. **Matching New User Session to Previous Profiles to Form a Collaborative Session:** The next step is to match a new user session to the existing profiles in order to determine close (or activated) profiles, and then combine the URLs in these profiles based on the degree of similarity between the new session and each originating profile to form a new collaborative-session,

4. **Query Formation and Scoring:** The final step transforms the collaborative-session into a content query as in the last step of Content-based filtering explained above.

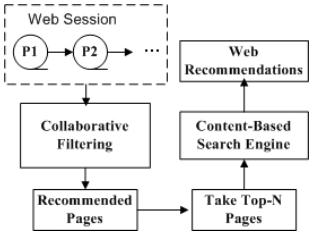
The entire transformation process can be summarized as follows:

session → close-profiles → collaborative-session → URLS → terms → query → results → re-rank results according to cosine similarity (result vector, query vector)

Note that just as in the case of *content-via-collaborative-filtering* described in Sec. 4.2.2.2, this approach will result in a *hybrid* (*collaborative and content*) recommender system, since the final recommendations will also take content into consideration. The collaborative ingredient comes from the fact that the content is not only formed from the current session, but also combines the content from a neighborhood of *"similar"* sessions. Also, this approach may be classified into the family of *cascaded hybrid recommendation strategies*, as well as *feature augmentation hybrid recommendations* [4].

We call this approach *content-based-via-collaborative-profile-filtering*, which is similar to [17], but it relies on profiles as a summary of all the previous sessions. This approach is also illustrated in Fig. 3.

Figure 3: *Cascaded Content-based-via-collaborative-profile-filtering* implemented via a Search Engine



4.2.2.4 Hybrid Content and Profile-based Collaborative Filtering with Weighted Combination.

The previously described Hybrid Content via Profile-based Collaborative Filtering works by *cascading* the collaborative filtering recommendations onto the content-based filtering to produce the final recommendations. Hence it works in *stages*. An obvious alternative would be to combine the collaborative filtering and content-based filtering *in parallel* instead of sequential manner. This approach falls within the family of *weighted hybrid recommendation strategies*, defined in [4] as methods where "*the scores (or votes) of several recommendation techniques are combined together to produce a single recommendation*". We call this approach *Weighted content-based-and-collaborative-profile-filtering*, and it is illustrated in Fig. 4. The recommendation procedure is performed using the following five steps:

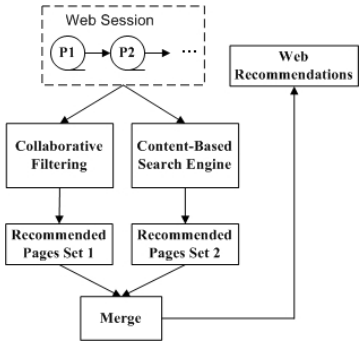
1. **Preliminary Mining of Usage Profiles (*done offline or online with scalable Web usage mining techniques*):** This step is identical to Step 1 in Sec. 4.2.2.3.
2. **Preliminary Crawling and Indexing of the Content (*done offline*):** This step is identical to Step 1 in Sec. 4.2.1.
3. **Matching New User Session to Previous Profiles to Form a Collaborative Session:** The next step is to match a new user session to the existing profiles in order to determine close (or activated) profiles, and then combine the URLs in these profiles based on the degree of similarity between the new session and each originating profile to form a new collaborative-session. This collaborative session with its inherent ranking of the URLs is then treated like a first component of the final recommendations (Recommended Set 1). the overall transformation process so far is as follows: **session → URLS → closest profiles → Recommended Set 1**
4. **Query Formation and Scoring:** this steps essentially consists of transforming the same new user session into a query that can be understood by the search engine, exactly as in pure content-based filtering. First, each URL in the user session is mapped to a set of terms that are most characteristic of this URLs, and then these terms are combined to form a query as in Fig. 1, i.e: **session → URLS → terms → query → Recommended Set 2**
5. **Final Collaborative and Content-based Recommendation Combination:** The final step consists of combining the two above recommended sets of items into a coherent list of recommendations. The main idea is to merge the two lists and sort them by the matching score assigned to each item. This transformation process can be denoted as follows:

Recommended Set 1 ⊕ Recommended Set 2 → Final Recommended Set

Note that we do not discuss the details of how the merging takes place, since many possible strategies can be implemented to do this. For example, one possible variation is to *weight* the scores of the recommendations in the two sets differently before sorting them into a final list, to give a bias toward the collaborative filtering component or the content-based filtering component depending on the application domain and on the performance of the recommendations.

Note that this approach will also result in a *hybrid* (*collaborative and content*) recommender system, since the final recommendations will take both similar users' interest (profiles) *and* content into consideration.

Figure 4: *Weighted Content-based-and-collaborative-profile-filtering* implemented via a Search Engine



4.2.3 Other Recommendation Approaches

The implementation of various other recommendation strategies can also be designed, although at first, the tweaking may not be as obvious. For example, it may be possible to tweak *rule-based*, *meta-content-attributes*, which are typically non-textual attributes that can be defined by use of *ontologies* (consider movie attributes for instance), user profile attributes such as demographic or declarative preferences, and *external business rules*, etc. Some of these different recommendation options are outlined below.

We can distinguish the following different cases where additional content or user profile attributes need to be incorporated into the recommendations:

Case 1: The content attribute is stored in a Database in the form of symbols, codes, etc (for instance on an e-commerce website this may include the *color* of a skirt such as *pink, blue, or black*, or the *type of heel* on a shoe such as *styletto, platform or flat* heel). In order to handle such attributes, the

code names can be added to the vocabulary of keywords, and may be treated as such in the item or underlying web page description. In this case, the procedure can be easily reduced to that of Content-based filtering as explained above. That is the viewed (or purchased) items can be mapped to a query and submitted to the search engine to return the most matching results. In cases where the attributes are tightly coupled with a specific item (for example the heel type is only attributable to a shoe and not a belt), then the session may be transformed into several separate queries that are submitted to the search engine sequentially, and the results combined. Note that this approach presupposes that the item Database has previously been converted to text and that its attributes have been stored into a reverse index that can make looking up a specific attribute easy and fast at recommendation time. An alternative to text transformation can be accomplished by storing the attributes as *fields* which are also indexable and searchable by Lucene (see Sec. 3.1.2).

Case 2: The content attributes are described or connected via an ontology or a set of rules: In this case, we can modify the *similarity* measure that is used by the search engine to match a query to existing content. Basically, the similarity measure should reflect all the relevant rules and ontological relationships between the different attributes and items to be compared in order to provide the recommendations. Note that here again, Lucene's ability to index and search *fielded* data can help deal with external attributes and simple rules (see Sec. 3.1.2).

Case 3: An elaborate user profile is available with non-textual attributes that describe a user, such as demographic attributes (age, income, gender, marital status, etc), or how a customer relates to certain known customer segments, such as (heavy spender, loyal customer, new customer, vulnerable customer, etc). This case can be solved by modifying the *similarity* matching so that all the rules that relate the recommended item contents to the user attributes are taken into account when performing the matching.

Case 4: A set of business rules are to be taken into account to recommend the items, which correspond to neither content-based nor collaborative filtering. This case is similar to the previous case in that the *similarity* matching module needs to be modified to implement these rules, so that a similarity degree may be suppressed or promoted depending on how a recommendation fits a given rule.

5. IMPLEMENTATION EXAMPLES

Our implementation started by crawling the web pages in the louisville.edu domain, where we limited the crawling to a depth of 6 levels resulting in 30,405 pages (this corresponds to Step 1 of content-based filtering). The content was crawled and indexed using nutch (23,097 unique terms were indexed), and the nutch search engine application was launched to accept queries and match them on the fly to the indexed content as they arrive. A proxy was set at one port on our server http://136.165.45.122:8089/ based on the Open Source SQUID Web proxy software (http://www.squid-cache.org/) and additional C code to track each session, convert it to an appropriate query, and submit this query to the nutch open source search engine to generate recommendations. Thus in order to view the recommendations which are available as the user navigates any of the web pages that have been crawled, the user must first set their browser to use http://136.165.45.122:8089/ as a proxy, and then continue their normal navigation in the web pages located within the domain of interest (in our case this was louisville.edu), where the *recommendations* link will be shown at the top of each page. The proxy approach offers a fast and non-invasive approach to add recommendations as was done in [2].

We are currently making progress on several recommendation strategies, including collaborative filtering methods, but have so far completed the simplest strategies which are the content-based filtering and anchor-based recommendations (all anchor terms of the sessions' browsed pages are merged into a Boolean query, and matched against the reverse index). In order to extract the content of a given URL, we used Nutch's built in functionalities for parsing various types of file extensions, and added a plugin for stop word elimination. The quality of *content-based filtering* recommendations was assessed by comparing the contents of the pages in the recommendation set to the pages to be visited in a real session, using the cosine similarity. The k most frequent terms in a page were used to form its vector space representation, First, we varied k to take the values: 5, 10, 20, 30, 40, and show the quality in Fig. 5. We noticed that the quality of the results saturated around $k = 30$, and thus do not show results for $k > 30$. The results used for evaluation purposes 450 sessions consisting of at least 3 clicks, and chosen randomly from one day's web logs. Then we fixed the maximum number of terms ($k = 10$) in the vector space representation and determined the quality for sessions of different lengths in Fig. 6.

Figure 5: Preliminary Results for Content-based filtering implemented via a Search Engine: Cosine similarity between Nth recommended page and to-be-visited pages for different maximum number of terms (k) used in vector space representation (forgetting factor F=0.5)

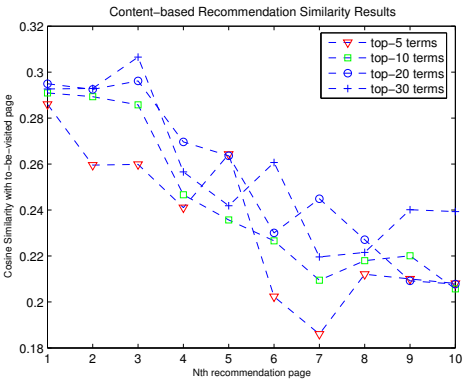


Figure 6: Preliminary Results for Content-based filtering implemented via a Search Engine: Cosine similarity between Nth recommended page and to-be-visited pages for different lengths of user sessions, and fixed maximum number of terms ($k = 10$) used in vector space representation (forgetting factor F=0.5)

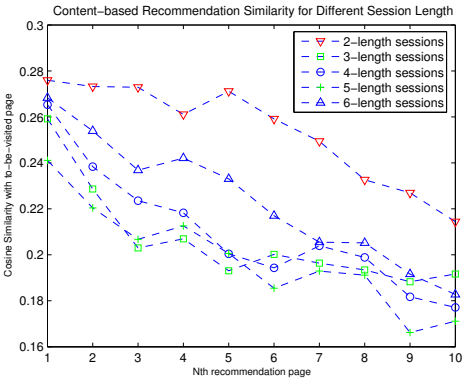


Table 1 shows that the time required for the content-based filtering recommendations, on a Linux server with Intel Xeon

Table 1: Average time to Recommendations for Content-Based Filtering vs. Number of Top Frequent Terms used in the Vector Space Representation

k	5	10	20	30	40
time in seconds	1.6	2.08	2.8	3.6	4.2

2.80GHz, 2CPU, and 2G memory, is modest (about 2 seconds for $k = 10$ top frequent terms used in the vector space representation), and that it grows almost linearly with the number of terms used in the vector space representation of the content (k).

6. CONCLUSIONS

In this paper, we have presented a systematic framework for a fast and easy implementation and deployment of a several types of recommendation strategies on one or more Websites, based on any available combination of open source tools that include crawling, indexing, and searching capabilities. There are many advantages of the Search Engine Tweaking Approach to Recommender System Design, which include: **(i)** the easy and *seamless integration* of multiple websites, such as affiliated e-commerce sites, even those that are hosted on disconnected web servers, **(ii)** *putting the user in control*: The proposed recommendation implementations are no longer confined to serve the interests of a particular website or business. In fact, the user (or a community of users) can install such a system on their own computers (or proxy in case of a community) to unify and dynamically link any group of websites that are relevant to their interests**, (iii)** the *Open Source advantage*: that offers the potential for improvements by the open source community, as well as providing and open platform for teaching and research**, (iv)** Inheriting from the *Information Retrieval legacy***,** including all the developments in Information Filtering, Personalized Information retrieval, and Query reformulation and expansion; **(v)** the *Semantic Web Vision*: Advances in information retrieval based on semantics can now also have an impact on recommender systems in a seamless manner.

6.1 Acknowledgments

This work is partially supported by National Science Foundation CAREER Award IIS-0133948 to O. Nasraoui.

7. REFERENCES

[1] Cdnow.com http://www.cdnow.com.

[2] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous Distributed Environments*, 1995.

[3] M. Balabanovic. An adaptive web page recommendation service. In *First International Conference on Autonomous Agents, Marinadel Rey, CA, 378-385*, 1997.

[4] R. Burke. Hybrid recommmender systems: Survey and experiments. In *In User Modeling and User-Adapted Interaction, 12(4):331-370*, 2002.

[5] R. Cooley, B. Mobasher, and J. Srivastava. Web mining: Information and pattern discovery on the world wide web. In *Proc. IEEE Intl. Conf. Tools with AI, Newport Beach, CA, pp. 558-567*, 1997.

[6] D. Cutting and J. Pedersen. Space optimizations for total ranking, riao (computer assisted ir) 1997. In *http://lucene.sf.net/papers/riao97.ps*.

[7] P. V. der Putten, J. N. Kok, and A. Gupta. Why the information explosion can be bad for data mining and how data fusion provides a way out. In *In Proc. of the 2nd SIAM International Conference on Data Mining*, 2002.

[8] X. Jin, J. C. French, and J. Michel. Saic & university of virginia at trec 2005: Hard track. In *In Proceedings of TREC 2005. On-line at http://trec.nist.gov*.

[9] X. Jin, J. C. French, and J. Michel. Saic & university of virginia at trec 2005: Hard track. In *Proc. 14th Text REtrieval Conference (TREC 2005), Gaithersburg, MD, November 15-18*, 2005.

[10] T. Joachims. Optimizing search engines using clickthrough data. In *In Proc. of the 8th ACM SIGKDD Conference, 133-142*, 2002.

[11] J. Konstan, B. M. J. Maltz, G. Herlocker, and J. Riedl. Grouplens: Collaborative filtering for usenet news. In *Communications of the ACM, March, p. 77-87*, 1997.

[12] G. Linden, B. Smith, and J. York. Amazon.com recommendations item-to-item collaborative filtering. In *IEEE Internet Computing*, volume 7.

[13] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Effective personalizaton based on association rule discovery from web usage data. In *ACM Workshop on Web information and data management, Atlanta, GA*, Nov 2001.

[14] O. Nasraoui, C. Cardona, C. Rojas, and F. Gonzalez. Mining evolving user profiles in noisy web clickstream data with a scalable immune system clustering algorithm. In *in Proc. of WebKDD 2003, Washington DC*, Aug 2003.

[15] O. Nasraoui, R. Krishnapuram, and A. Joshi. Mining web access logs using a relational clustering algorithm based on a robust estimator. In *8th International World Wide Web Conference, Toronto, pp. 40-41*, 1999.

[16] O. Nasraoui, R. Krishnapuram, A. Joshi, and T. Kamdar. Automatic web user profiling and personalization using robust fuzzy relational clustering. In *in E-Commerce and Intelligent Methods in the series Studies in Fuzziness and Soft Computing, J.Segovia, P. Szczepaniak, and M. Niedzwiedzinski, Ed, Springer-Verlag*, 2002.

[17] M. Pazzani. A framework for collaborative. In *Content-Based and Demographic Filtering, AI Review, 13(5-6):393-408*, 1999.

[18] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. In *Machine Learning, 27:313331*, 1997.

[19] M. Perkowitz and O. Etzioni. Adaptive web sites: Automatically learning for user access patterns. In *Proc. 6th int. WWW conference*, 1997.

[20] B. M. Sarwar, J. A. Konstan, A. Borchers, J. Herlocker, B. Miller, and J. Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *In Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work, Seattle, Washington*, 1998.

[21] J. Schafer, J. Konstan, and J. Reidel. Recommender systems in e-commerce. In *Proc. ACM Conf. E-commerce*, 1999.

[22] M. Spiliopoulou and L. C. Faulstich. Wum:a web utilization miner. In *in Proceedings of EDBT workshop WebDB98, Valencia, Spain*, 1999.

[23] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. Web usage mining: Discovery and applications of usage patterns from web data. In *SIGKDD Explorations*, volume 1, Jan 2000.

[24] L. Terveen, W. Hill, B. Amento, D. McDonald, and J. Creter. Phoaks: A system for sharing recommendations. In *Communications of the ACM, 40(3), 59-62*, 1997.

An Effectiveness Measure for Evaluating Open Retrieval Systems*

Hsieh-Chang Tu and Jieh Hsiang
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan
{tu,hsiang}@csie.ntu.edu.tw

ABSTRACT

One may regard Web *search engines* as information retrieval (IR) systems over an *open* repository that contains an indefinite number of Web pages. Due to the open nature of the Web, traditional effectiveness measures such as *recall-precision curves* and *MAP* (mean average precision) becomes questionable. In this paper, we demonstrate how a misvalued recall can mislead evaluation. Then, we define a measure called *relnum-precision plot* that does not require a priori knowledge of the total number of relevant documents. We propose to use *average relnum-precision plot*, or its single-valued derivations AP@n and AP#N, to evaluate open systems.

1. INTRODUCTION

Information retrieval (IR) is a task to find documents from a repository. The emergence of the WWW (World Wide Web) was a watershed event for information retrieval. Regarding a Web page as a document, one may consider all accessible Web pages to be a huge document repository.

This repository, however, has some properties that distinguishes it from a traditional IR repository. Traditional IR assumes, though often not mentioned, that systems retrieve documents over a *closed* document space. This assumption is not valid for the Web, since the space is *open*, in the sense that it contains a large amount of sparsely located documents, so distributed and massive in quantity that these properties make the space boundary unclear. By a clear boundary, we mean that one can determine the exact space size, or at least confidently estimate this value to be within a small range, with relatively low cost.

Conceptually one could claim to have a closed repository by "freezing" the entire Web at a fixed time. However, it is not possible to get such a snapshot in reality. There is a tremendous amount – theoretically an unbounded number – of Web documents available via valid URLs, and its content changes on a daily basis.

An IR system typically accepts a string of words as a *query*, and returns a ranked list of documents from that repository. Retrieval effectiveness is then evaluated by assessing whether documents in the result list are *relevant* to the *search request* [4, 5, 7]. A system that finds more relevant documents and less not relevant ones is regarded as better. Among effectiveness measures, *precision* and *recall* (or their variations) are the most common [5, 7, 8].

The main problem of recall is that it depends on a priori knowledge of the total *relnum* (number of relevant documents). Since *openness* implies incompleteness, it is not feasible to collect all the relevant documents (except some cases that one has known the relevant documents in advance). Being unable to obtain all relevant documents leads to a misvalued – often underestimated – total relnum. A misvalued total relnum results in a misvalued recall, which in turn challenges the reliability of evaluation. This makes the use of recall-based measures questionable in evaluating Web IR systems.

We organize this paper as follow. After a short introduction on definitions and notations, we give examples that demonstrates potential pitfalls of a misvalued recall. Then, in Section 4, we propose a measure to evaluate open systems.

2. DEFINITIONS AND NOTATIONS

Let Δ be the document space. We use $\mathcal{R}_\tau \subseteq \Delta$ to denote the set of documents relevant to search request τ. Given a query q, let $L(q,n)$ denote the top n ranked documents returned by the system. We call n the *document cutoff value* (DCV). We define $L(q,0) = \emptyset$, the empty set. If the number of returned documents is m, then $L(q,n+1) = L(q,n)$ for $n \geq m$. We then denote the set of all returned documents by $L(q) = \bigcup_{n=1}^{\infty} L(q,n)$. We denote the complement of a set S by $\overline{S} = \Delta - S$, and use $|S|$ to denote the number of elements in S.

Intuitively, a system is more effective if its return is closer to what a user wants. In other words, assuming that all relevant documents are equally desirable, a system performs

*This research is supported in part by grant NSC94-2422-H-002-008 of the National Science Council of the Republic of China.

OSIR '06 Seattle, USA

better if its top n ranked documents $L(q,n)$ approximates the relevant set $\mathcal{R}_\tau$ more. The measure *precision* computes the proportion of $L(q,n)$[1] to be in $\mathcal{R}_\tau$, while recall evaluates ratio of $L(q,n)$ to $\mathcal{R}_\tau$. Let $a = |L(q,n) \cap \mathcal{R}_\tau|$, $b = |L(q,n) \cap \overline{\mathcal{R}_\tau}|$, $c = |\overline{L(q,n)} \cap \mathcal{R}_\tau|$, $d = |\overline{L(q,n)} \cap \overline{\mathcal{R}_\tau}|$ be the number of documents which are retrieved and relevant, retrieved but not relevant, relevant but not retrieved, and neither retrieved nor relevant respectively. *Precision* and *recall* are defined to be

$$precision = \frac{a}{a+b}, \quad (1)$$

$$recall = \frac{a}{a+c}. \quad (2)$$

Notice that Definition (2) only defines recall on discrete points. For instance, the precision at recall level 0.1 may be undefined if the total relnum equals 11. This makes difficult the computation of averaged performance, since a recall level can be undefined for some requests. Several interpolating methods have been proposed to fill the undefined region [5, 7, 8]. The overall system effectiveness is then represented by a *recall-precision curve* [7], which is a two-dimensional graph that contains points $\{(r,p)\}$ where $0 \le r \le 1$ is a recall level and p is the precision after attaining this recall.

3. POTENTIAL PITFALLS OF MISVALUED RECALL

Being unable to collect all relevant documents leads to a misvalued – often underestimated – total relnum $|\mathcal{R}_\tau|$. A misvalued total relnum results in a misvalued recall. In the following, we show how a misvalued recall can reverse the result of an evaluation.

EXAMPLE 1 (MISVALUED RECALL OVER ONE QUERY). *Suppose one compares two retrieval systems S_1 and S_2 by recall-precision curves. Given a query, let R be the actual number of relevant documents and let r be its estimate (obtained by pooling [2, 3, 9] or other sampling techniques). If $r < R \le 2r$, then comparing*

- *precision at estimated recall level $\frac{1}{r}$, and*
- *precision at actual recall level $\frac{1}{r}$,*

can lead to different conclusions.

Details: We give an example to demonstrate this possibility. Consider the following tables[2] that list documents returned by S_1 and S_2 respectively. We use the symbol '$\surd$' to denote a relevant document, and mark a not relevant one by '$\times$.' We assume that the top four documents returned by S_1 are d_1, d_2, d_3 and d_4, and those returned by S_2 are d_2, d_1, d_4, and d_3. Documents d_1 and d_4 are assessed as relevant, while d_2 and d_3 are not relevant.

One possible choice of R and r is to let $R = 5$ and $r = 4$. In such a case, the corresponding recall levels will be $\frac{1}{r} = 25\%$ and $\frac{1}{R} = 20\%$.

S_1:

DCV	1	2	3	4	...
document id	d_1	d_2	d_3	d_4	...
assessment	$\surd$	$\times$	$\times$	$\surd$	...
precision	100%			50%	...
estimated recall	$\frac{1}{r}$			$\frac{2}{r}$	...
actual recall	$\frac{1}{R}$	$\frac{1}{R}$	$\frac{1}{R} < \frac{1}{r}$	$\frac{2}{R} \ge \frac{1}{r}$	...

S_2:

DCV	1	2	3	4	...
document id	d_2	d_1	d_4	d_3	...
assessment	$\times$	$\surd$	$\surd$	$\times$	...
precision		50%	66.67%		...
estimated recall		$\frac{1}{r}$	$\frac{2}{r}$		...
actual recall		$\frac{1}{R} < \frac{1}{r}$	$\frac{2}{R} \ge \frac{1}{r}$		...

Now we compare effectiveness at recall $\frac{1}{r}$. The rows marked "actual recall" say that S_1 achieves recall $\frac{1}{r}$ with precision $\frac{1}{2} = 50\%$ (at $DCV = 4$), and S_2 reaches the same recall with precision $\frac{2}{3} = 66.67\%$ (at $DCV = 3$). It follows that S_2 actually performs better.

On the other hand, suppose we evaluate systems by "estimated recall." The above tables say that S_1 achieves recall $\frac{1}{r}$ with precision 100% (at $DCV = 1$), and S_2 attains the same recall with precision 50% (at $DCV = 2$). Thus S_1 performs better than S_2. This conclusion disagrees with the above observation that S_2 actually performs better. □

The reason for this phenomenon is the following: Effectiveness of systems is compared over a fixed recall level $\frac{1}{R}$ which assumes a total relnum R. When the total relnum is underestimated, such a comparison is actually taken on recall $\frac{1}{r}$, where $r < R$. An undesirable result occurs when a system (e.g., S_1 in the above example) achieves better precision than the other (i.e., S_2) at recall $\frac{1}{R}$ but yields worse precision at recall $\frac{1}{r}$.

The measure *mean average precision* (MAP) is a single measure most often used in IR research to represent the overall effectiveness performance of a system [1, 6, 8]. This measure is defined to be the precision averaged over *all* relevant documents for a request. Equivalently, MAP is the precision averaged over all (non-interpolated) recall levels defined by Definition (2). The MAP over many requests is defined to be the average of MAPs for each request. One might wonder whether an underestimated total relnum may result in a MAP that changes evaluation results. The answer is Yes, if we evaluate systems over several requests. As we shall shown in Example 2, an underestimated total relnum can result in an interesting phenomenon, where a system having a higher precision at *any* recall level should be regarded as one that performs worse.

[1] If one wants to talk about precision of the whole retrieved documents $L(q) = \bigcup_n L(q,n)$, simply let $n \to \infty$ and $\lim_{n\to\infty} L(q,n) = L(q)$.

[2] Here we adopt the following interpolation method to define the precision at a recall level not applicable by Definition (2) [7]. Many DCVs may yield the same number of relevant documents. Let n be the smallest DCV that yields a recall level greater than or equal to r. We define the precision at recall level r to be the precision under $DCV = n$. Thus, in this example, system S_1 achieves (actual) recall $\frac{1}{r}$ at $DCV = 4$, which is the smallest DCV that yields recall greater than or equal to $\frac{1}{r}$.

Our example requires recall to be defined on all values between zero and one. We define the precision at recall level zero to be one, and adopt a *linear interpolation* technique to cover the undefined region between two defined points [5, 7]. In other words, if a recall level $0 < r < 1$ does not exist, we define the point (r, p) by interpolating its two nearest neighbors (r_1, p_1) and (r_2, p_2). That is,

$$(r,\ p) = \left(r,\ p_1 + \frac{r - r_1}{r_2 - r_1}(p_2 - p_1)\right).$$

One may regard a recall-precision curve as a function $f(x)$, where x represents a recall level and $f(x)$ is the corresponding precision. Let $f_i(x)$ represent the recall-precision curve of system S_i. We say that system S_1 performs *uniformly better* than S_2 (denoted by $S_1 \succ S_2$) iff for *all* $x > 0$, $f_1(x) > f_2(x)$ (denoted by $f_1(x) \rhd f_2(x)$). Intuitively, a system performs uniformly better means that it yields higher precision at all recall levels.

When the total relnum is not exact but an estimation, the corresponding recall is not exact neither. It turns out that a recall-precision curve becomes an "estimated recall"-precision one. In the following, we give a sufficient condition under which an exact total relnum and its estimate can result in different conclusions.

PROPOSITION 1. *Suppose that one wants to evaluate the performance of two systems S_1 and S_2 by sample queries q_1 and q_2. Let R_1, R_2 be the total relnums of query q_1 and q_2, and let r_1, r_2 be estimations of R_1, R_2 respectively. Let $\phi_{ij}(x)$ denote the actual recall-precision curve of system S_i on query q_j ($i, j \in \{1, 2\}$). If, for $0 < x \leq 1$,*

1. *$R_1 = r_1$ and $R_2 = \rho r_2$, for some $\rho > 1$,*
2. *$\phi_{11}(x) + \phi_{12}(x) < \phi_{21}(x) + \phi_{22}(x)$, and*
3. *$\phi_{11}(x) + \phi_{12}(\frac{x}{\rho}) > \phi_{21}(x) + \phi_{22}(\frac{x}{\rho})$,*

then

- *$S_1 \succ S_2$ under the average "estimated recall"-precision curves, but*
- *$S_2 \succ S_1$ under the (actual) average recall-precision curves.*

Proof: The "actual performance" of system S_i is computed by averaging $\phi_{i1}(x)$ and $\phi_{i2}(x)$:

$$\phi_i(x) = \frac{1}{2}\left(\phi_{i1}(x) + \phi_{i2}(x)\right).$$

It is clear that $S_2 \succ S_1$, since for all $0 < x \leq 1$,

$$\phi_1(x) = \frac{1}{2}(\phi_{11}(x) + \phi_{12}(x)) < \frac{1}{2}(\phi_{21}(x) + \phi_{22}(x)) = \phi_2(x).$$

On the other hand, we denote by $\psi_{ij}(x)$ the "estimated recall"-precision curve of system S_i acting on the query q_j. The difference of $\phi_{ij}(x)$ to $\psi_{ij}(x)$ is that the former curve has an exact total relnum, while the latter uses an estimate one. The "estimated" overall performance of system S_i is

$$\psi_i(x) = \frac{1}{2}\left(\psi_{i1}(x) + \psi_{i2}(x)\right).$$

A $\rho > 1$ indicates that the total relnum is underestimated. If $\rho = 2$, then the precision at 50% "estimated recall" is actually the precision at 25% actual recall. Thus an "estimated recall"-precision curve $\psi(x)$ is simply an initial segment of the actual recall-precision one, i.e., $\psi(x) = \phi(\frac{x}{\rho})$. Since $r_1 = R_1$ and $r_2 = \frac{1}{\rho}R_2$, it follows that

$$\begin{aligned} \psi_{i1}(x) &= \phi_{i1}(x), \\ \psi_{i2}(x) &= \phi_{i2}(\frac{x}{\rho}) \end{aligned}$$

for $0 < x \leq 1$. Thus

$$\begin{aligned} \psi_1(x) &= \frac{1}{2}(\psi_{11}(x) + \psi_{12}(x)) = \frac{1}{2}\left(\phi_{11}(x) + \phi_{12}(\frac{x}{\rho})\right) \\ &> \frac{1}{2}\left(\phi_{21}(x) + \phi_{22}(\frac{x}{\rho})\right) = \frac{1}{2}(\psi_{21}(x) + \psi_{22}(x)) \\ &= \psi_2(x) \end{aligned}$$

for $0 < x \leq 1$. That is, $S_1 \succ S_2$ under the average "estimated recall"-precision curves. □

Next we present an example that satisfies the above proposition. Specifically, we choose $\rho = 2$, $R_1 = r_1 = 2$, $R_2 = 4$ and $r_2 = \frac{R_2}{\rho} = 2$.

EXAMPLE 2 (MISVALUED RECALL CAN MISLEAD). *Let $\phi_1(x)$, $\phi_2(x)$ be the average recall-precision curves of two systems S_1, S_2 respectively. If the total relnum is not exact but an estimated value, instead we get average "estimated recall"-precision curves $\psi_1(x)$ and $\psi_2(x)$. It is possible that $\psi_1(x) \rhd \psi_2(x)$, but $\phi_2(x) \rhd \phi_1(x)$.*

Details: Suppose we have a repository of more than 200 documents. We want to evaluate two systems S_1 and S_2, over two sample requests τ_1 and τ_2. Let $|\mathcal{R}_{\tau_1}| = 2$ and $|\mathcal{R}_{\tau_2}| = 4$. That is, there are two documents relevant to request τ_1, and four documents relevant to τ_2. We also assume that one underestimates the total relnum to be two for τ_2. Let q_1 and q_2 be the queries for τ_1 and τ_2 respectively. Consider the tables shown in Table 1. Each table T_{ij} ($i, j \in \{1, 2\}$) gives the return list of system S_i on query q_j.

Notice that both systems return at least 200 documents to these queries. Since the total relnum for τ_2 is 4 but underestimated to be 2, there are two documents relevant to τ_2 but regarded as not relevant. They are marked by '$\circ$' in Table 1.

Figure 1 illustrates the recall-precision curves. The symbol $\phi_{ij}(x)$ ($\psi_{ij}(x)$) denotes the actual (estimated) recall-precision curve of system S_i on query q_j ($i, j \in \{1, 2\}$). From the subfigure (f) of Figure 1, since $\psi_1(x) \rhd \psi_2(x)$, one concludes that $S_1 \succ S2$ by the average "estimated recall"-precision curves. However, subfigure (c) says that $\phi_2(x) \rhd \phi_1(x)$, which means $S_2 \succ S_1$ by the (actual) average

T_{11}:

DCV	1	2	3	4	5	6	7	8	9	...	200	...
assessment	×	√	×	×	×	√	×	×	×	×	×	×
precision		1/2				1/3						
(actual/estimated) recall		50%				100%						

T_{21}:

DCV	1	2	3	4	5	...	99	100	101	...	200	...
assessment	×	×	×	√	×	...	×	√	×	×	×	×
precision				1/4				1/50				
(actual/estimated) recall				50%				100%				

T_{12}:

DCV	1	2	3	4	...	9	10	11	...	198	199	200	...
assessment	×	×	√	×	×	×	√	×	×	×	○	○	×
precision			1/3				1/5				3/199	1/50	
actual recall			25%				50%				75%	100%	
estimated recall			50%				100%						

T_{22}:

DCV	1	2	3	4	5	6	7	...	200	...
assessment	×	√	×	√	○	○	×	×	×	×
precision		1/2		1/2	3/5	2/3				
actual recall		25%		50%	75%	100%				
estimated recall		50%		100%						

Table 1: Comparing effectiveness over two systems by two queries. Table T_{ij} shows the retrieved result of system S_i on query q_j. The symbol '√' denotes a relevant document, and '×' marks a not relevant one. The symbol '○' represents a document which is actually relevant but unfortunately regarded as not relevant. In this example, there are two such documents for query q_2 ($|\mathcal{R}_{\tau_2}|$ is underestimated).

recall-precision curves. In other words, the underestimated total relnum for τ_2 results in an incorrect conclusion that S_1 performs better.

Remarks: The measure MAP is also unreliable in this example, in the sense that the comparison of systems according to an underestimated total relnum differs from that according to a correct one. (This is because a system performs uniformly better always yields a larger MAP.) Recall that the MAP for a request is defined to be the average of precisions over all relevant documents. The average precisions for T_{11}, T_{12}, T_{21}, T_{22} are $\frac{1}{2}(\frac{1}{2}+\frac{1}{3}) = 0.417$, $\frac{1}{2}(\frac{1}{3}+\frac{1}{5}) = 0.267$, $\frac{1}{2}(\frac{1}{4}+\frac{1}{50}) = 0.135$, and $\frac{1}{2}(\frac{1}{2}+\frac{1}{2}) = 0.5$ respectively, if the total relnum for τ_2 is underestimated. Since the MAP is $\frac{1}{2}(0.417+0.267) = 0.342$ for system S_1 and is $\frac{1}{2}(0.135+0.5) = 0.318$ for S_2, one gets a conclusion that S_1 performs better. On the other hand, if the total relnum is exact, the average precisions for T_{11}, T_{12}, T_{21}, T_{22} are 0.417, $\frac{1}{4}(\frac{1}{3}+\frac{1}{5}+\frac{3}{199}+\frac{1}{50}) = 0.142$, 0.135, and $\frac{1}{4}(\frac{1}{2}+\frac{1}{2}+\frac{3}{5}+\frac{2}{3}) = 0.567$. Thus the MAPs for S_1 and S_2 are $\frac{1}{2}(0.417+0.142) = 0.279$ and $\frac{1}{2}(0.135+0.567) = 0.351$ respectively, which means S_2 actually performs better in this case. □

The above example shows that people can be fooled by an "estimated recall"-precision curve. Notice that only one of the two queries, namely q_2, is 50% underestimated. That is, a single misvalued total relnum can reverse the result of an evaluation. Unless one can be certain of the total relnum, it is theoretically flawed to evaluate a system by an average "estimated recall"-precision curve or its variations. The probability to underestimate a total relnum by 50% could be low for a closed repository, but certainly one cannot precisely compute recall in an open space. Therefore, it is not reasonable to use recall-based measures in evaluating open systems.

4. A PROPOSED MEASURE

In this section we propose a measure, called *relnum-precision plot*, to evaluate open systems. We shall examine its intuitive interpretation, and discuss its relationship to a measure adopted in the TREC (Text REtrieval Conference) Web track.

4.1 Relnum-Precision Plot

Recall that $L(q, n)$ denotes the top n ranked documents returned for query q, and $\mathcal{R}_\tau$ represents the set of documents relevant to search request τ. Given a request τ, we define the *precision (of a query q) after retrieving n documents* to be

$$p_{\tau,q}(n) \;=\; \frac{|L(q,\ n) \cap \mathcal{R}_\tau|}{n}. \tag{3}$$

We then define the *precision (of a query q) after retrieving n relevant documents* by

$$\pi_{\tau,q}(n) \;=\; p_{\tau,q}(t), \tag{4}$$

where t is the smallest number such that $|L(q,\ t) \cap \mathcal{R}_\tau| = n$. If there is no such a t, which means that q retrieves less than n relevant documents, then we define $\pi_{\tau,q}(n)$ to be zero.

A *relnum-precision plot* $\mathcal{P}_{\tau,q}\#N$ is a set of points $(n, \pi_{\tau,q}(n))$, where n ($\leq N$) denotes the number of relevant documents retrieved so far. We call N the *relnum bound*. It limits the maximal number of relevant documents one cares about. We denote $\mathcal{P}_\tau\#N$ to be the relnum-precision plot $\mathcal{P}_{\tau,q}\#N$ such that q is the query one chooses as the "best" in evaluation.

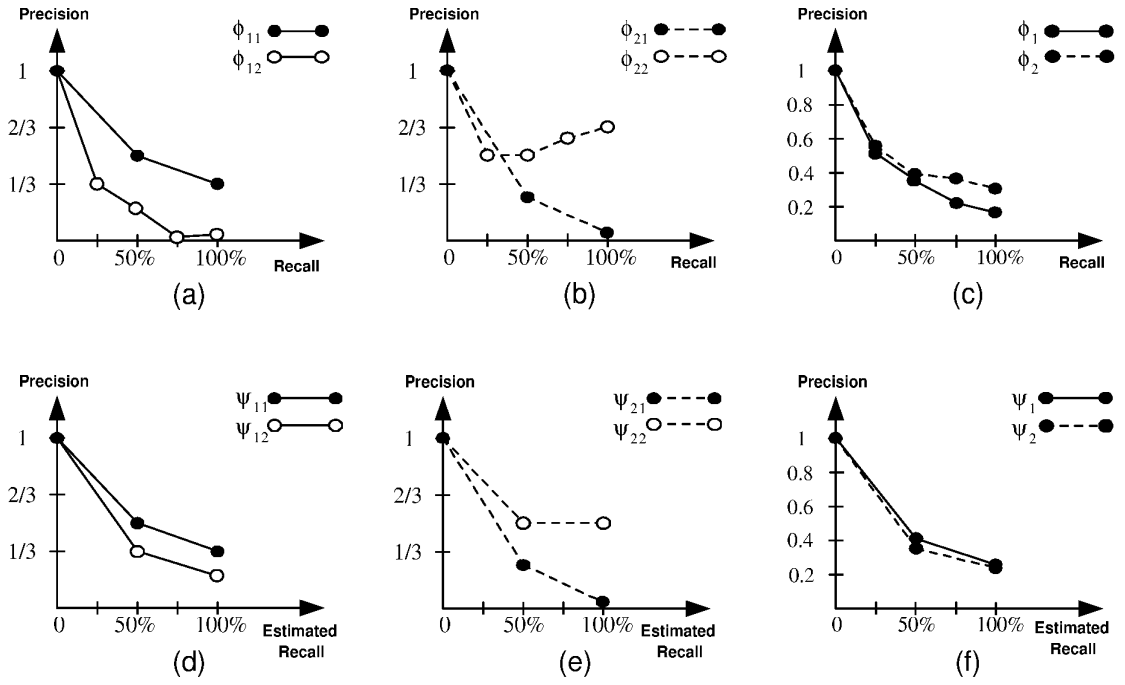


Figure 1: Performance curves which illustrate how a misvalued recall can fool an evaluation. All curves are drawn from Table 1 with linear interpolation. (a) The recall-precision curves ϕ_{11} and ϕ_{12}. (b) The recall-precision curves ϕ_{21} and ϕ_{22}. (c) The average recall-precision curves ϕ_1 and ϕ_2. (d) The "estimated recall"-precision curves ψ_{11} and ψ_{12}. (e) The "estimated recall"-precision curves ψ_{21} and ψ_{22}. (f) The average "estimated recall"-precision curves ψ_1 and ψ_2.

We model the overall system effectiveness by an *average relnum-precision plot*, which averages performance over many individual plots. In particular, let $\mathcal{T}$ be the set of search requests that one evaluates over, we define the *average relnum-precision plot* $\mathcal{P}_{\mathcal{T}}\#N$ to be a collection of points

$$(n,\, \pi_{\mathcal{T}}(n)) = \left(n,\, \frac{\sum_{\tau\in\mathcal{T}} \pi_\tau(n)}{|\mathcal{T}|}\right), \qquad (5)$$

where $1 \le n \le N$, $(n, \pi_\tau(n)) \in \mathcal{P}_\tau\#N$, and $\frac{\sum_{\tau\in\mathcal{T}} \pi_\tau(n)}{|\mathcal{T}|}$ takes the *arithmetic mean* of $\pi_\tau(n)$ over all requests $\tau \in \mathcal{T}$.

Given a fixed relnum bound N, the average relnum-precision plot $\mathcal{P}_{\mathcal{T}}\#N$ contains N points. In practice, it is often convenient to devise a single-valued measure that indexes the overall system performance. Implicitly assuming a fixed $\mathcal{T}$, we denote $\pi_{\mathcal{T}}(n)$ by AP@n (Average of Precisions at *relnum* $= n$), and define AP#N (Average of Precisions in the average relnum-precision plot with relnum bound N) to be

$$\mathrm{AP}\#N = \frac{\sum_{n=1}^{N} \mathrm{AP}@n}{|\mathcal{T}|} = \frac{\sum_{n=1}^{N} \pi_{\mathcal{T}}(n)}{|\mathcal{T}|}. \qquad (6)$$

We propose to use an average relnum-precision plot, or its derivations AP@n and AP#N, to evaluate open systems.

EXAMPLE 3 (COMPUTING $\mathcal{P}_{\mathcal{T}}\#3$ AND AP#3). *This example illustrates a computation of $\mathcal{P}_{\mathcal{T}}\#3$ and AP#3, where $\mathcal{T} = \{\sigma, \tau\}$.*

Details: The following tables, headed by symbols $q(x)$, show some document assessments of $L(q(x))$ to request x:

$q(\sigma)$:

DCV	1	2	3	4	5	6	...
assessment	×	√	√	×	√	×	...

$q(\tau)$:

DCV	1	2	3	4	⊥
assessment	√	×	×	√	⊥

We first compute $p_{\sigma,q(\sigma)}(n)$ and $p_{\tau,q(\tau)}(n)$ as follow:

n	1	2	3	4	5	6	...
$p_{\sigma,q(\sigma)}(n)$	0	1/2	2/3	2/4	3/5	3/6	...
$p_{\tau,q(\tau)}(n)$	1	1/2	1/3	2/4	2/5	2/6	...

Then, by Definition 4, we obtain $\pi_{\sigma,q(\sigma)}(n)$ and $\pi_{\tau,q(\tau)}(n)$, where $1 \le n \le 3$:

n	1	2	3
$\pi_{\sigma,q(\sigma)}(n)$	1/2	2/3	3/5
$\pi_{\tau,q(\tau)}(n)$	1	2/4	0

Computing $\pi_{\mathcal{T}}(n) = \frac{1}{2}(\pi_{\sigma,q(\sigma)}(n) + \pi_{\tau,q(\tau)}(n))$ yields $\mathcal{P}_{\mathcal{T}}\#3$, which equals $\{(n, \pi_{\mathcal{T}}(n)) : 1 \le n \le 3\}$:

n	1	2	3
$\pi_{\mathcal{T}}(n)$	3/4	7/12	3/10

That is, AP@1 = $\frac{3}{4}$, AP@2 = $\frac{7}{12}$, and AP@3 = $\frac{3}{10}$. Therefore, AP#3 = $\frac{1}{3}(\frac{3}{4} + \frac{7}{12} + \frac{3}{10}) = 0.544$.

Remarks:

- In this example, the system returns more than 6 documents for query q_σ ($|L(q_\sigma)| \geq 6$) and exactly 4 documents for query q_τ ($|L(q_\tau)| = 4$). We know that $|\mathcal{R}_\sigma| \geq 3$ and $|\mathcal{R}_\tau| \geq 2$, but we do not know $|\mathcal{R}_\sigma|$ and $|\mathcal{R}_\tau|$. The computation of a relnum-precision plot $\mathcal{P}_\tau\#N$ requires $|L(q,n) \cap \mathcal{R}_\tau|$ but not $|\mathcal{R}_\tau|$. From this point of view, we may regard a relnum-precision plot as a kind of partial measures that do not require total knowledge about all the relevant documents.

- Since an open IR system (i.e., one over an open repository) may return much more documents than what a real user wants, in practice we may set a *cutoff bound* M such that all the documents ranked after M (i.e., those in the set difference $L(q) - L(q,M)$) are regarded as not relevant.

□

4.2 Interpretation

Points on $\mathcal{P}_\mathcal{T}\#N$ have an intuitive meaning. Let $(x, \pi_\mathcal{T}(n))$ be a point on $\mathcal{P}_\mathcal{T}\#N$. By Definition (5) we have

$$\pi_\mathcal{T}(n) = \frac{\sum_{\tau\in\mathcal{T}} \pi_\tau(n)}{|\mathcal{T}|} = n \cdot \frac{\sum_{\tau\in\mathcal{T}} v_\tau^{-1}(n)}{|\mathcal{T}|} = n \cdot H^{-1},$$

where $v_\tau(n) = n \cdot (\pi_\tau(n))^{-1}$ is the *number of retrieved documents required to obtain n relevant ones*, and $H = |\mathcal{T}| \cdot \left(\sum_{\tau\in\mathcal{T}} v_\tau^{-1}(n)\right)^{-1}$ is the *harmonic mean* over the set $\{v_\tau(n)\}_{\tau\in\mathcal{T}}$[3]. In other words, the above equation says that we can interpret $(n, \pi_\mathcal{T}(n))$ as follow: To obtain n relevant documents, in harmonic average (over search requests) one needs to retrieve $H = n \cdot (\pi_\mathcal{T}(n))^{-1}$ documents.

In Example 3, for instance, it is easy to verify that AP@3 = $\pi_\mathcal{T}(3)$ is the harmonic mean of $v_{\sigma,q(\sigma)}(3) = \frac{3}{\pi_{\sigma,q(\sigma)}(3)} = 5$ and $v_{\tau,q(\tau)}(3) = \frac{3}{\pi_{\tau,q(\tau)}(3)} = \infty$:

$$\pi_\mathcal{T}(3) = \frac{3}{10} = \frac{2}{\frac{3}{5} + \frac{3}{\infty}}.$$

That is, one needs to retrieve 5 documents by $q(\sigma)$ to obtain 3 relevant documents, and fails to find so many relevant ones by $q(\tau)$. In other words, in order to retrieve 3 relevant documents, in harmonic mean one has to retrieve $\frac{2}{5^{-1}+\infty^{-1}} = 10$, which equals $H = \frac{3}{\pi_\mathcal{T}(3)}$, documents.

4.3 Discussion

A special case of $N = 1$ results in AP#1 = $\pi_\mathcal{T}(1)$, a measure called MRR1 (mean reciprocal rank of first correct answer). MRR1 is one of the four measures adopted in the TREC Web track [8, p. 201].

[3]The *harmonic mean* of n numbers x_1, ..., x_n is defined to be $\frac{n}{\frac{1}{x_1}+\ldots+\frac{1}{x_n}}$.

The other three measures adopted in the TREC Web track are $P@n$ (the precision at a fixed document cutoff n), $S@n$ (proportion of queries for which a correct answer is within the top n), and MAP. As we discussed in Section 3, it is questionable to use MAP to evaluate an open system. In contrast, an average relnum-precision plot (or its derivations AP@n and AP#N) is more practical, in the sense that it does not require a priori knowledge of total relnum.

5. CONCLUSION

A repository is *open* if its space boundary is not clear. The Web space is open, since it contains an indefinite number of Web documents. By nature, openness makes it not feasible to estimate precisely the total *relnum* (number of relevant documents).

In this paper, we demonstrate how an underestimated total relnum can result in a misvalued recall, which in turn can make an evaluation unreliable. Since the Web space is open by nature, it is questionable to adopt recall-based measures to evaluate Web IR systems.

With a set-theoretic approach, we define a measure called *relnum-precision plot*. We propose to use *average relnum-precision plot*, or its single-valued derivations AP@n and AP#N, to evaluate open systems.

6. REFERENCES

[1] C. Buckley and E. M. Voorhees. Retrieval evaluation with incomplete information. *ACM SIGIR'04*, pages 25–32, 2004.

[2] D. K. Harman. Overview of the first trec conference. *ACM SIGIR'93*, pages 36–47, 1993.

[3] D. K. Harman. Overview of the second text retrieval conference (trec-2). *Information Processing & Management*, 31(3):271–289, 1995.

[4] S. Mizzaro. Relevance: The whole history. *Journal of the American Society for Information Science*, 48(9):810–832, 1997.

[5] G. Salton, editor. *The SMART Retrieval System: Experiments in Automatic Document Processing.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1971.

[6] M. Sanderson and J. Zobel. Information retrieval system evaluation: effort, sensitivity, and reliability. *ACM SIGIR'05*, pages 162–169, 2005.

[7] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, 2 edition, 1979.

[8] E. M. Voorhees and D. K. Harman. *TREC: Experiment and Evalluation in Information Retrieval.* The MIT Press, Cambridge, Massachusetts, 2005.

[9] P. Wallis and J. A. Thom. Relevance judgements for assessing recall. *Information Processing & Management*, 32(3):273–286, 1996.

SHOW AND TELL: A Seamlessly Integrated Tool For Searching with Image Content And Text

Zhiyong Zhang Carlos Rojas Olfa Nasraoui
Hichem Frigui
Department of Computer Engineering and Computer Science
University of Louisville
Speed School of Engineering, Louisville, KY 40292
{olfa.nasraoui}@louisville.edu

ABSTRACT

In this paper, an image search tool that combines keyword and image content feature *querying* and *search* is presented. The developed search tool tries to bridge the gap between *commercial search engines*, which are based on *keyword* search, and *CBIR* (Content Based Image Retrieval) systems developed mostly in the academic field, designed to search based on image content. The tool is implemented by building on and extending the open source text-based search engine *Nutch* and its powerful *Lucene* based crawling and indexing capabilities. Several user friendly search options are provided to allow users to query the index using not only words, but also by showing an image example, as well as image feature descriptions. Even though we evaluate the developed tool by running a set of controlled experiments on the *COREL' 5000* image database, the developed search tool is able to crawl images from the World Wide Web at a larger scale.

1. INTRODUCTION

Current large scale image search engines like Google , Yahoo, Altavista, etc. base their image search capabilities mainly on text features associated to pictures, such as file name, image URL, tags, etc, and, marginally, on criteria such as *image size, color vs. "Black and White" images*, or *image format* (such as JPEG, GIF, etc). Even specialized online stock images *Corbis* (http://corbis .com/), *Getty* (http://gettyimages.com/), and *images.com* are also fundamentally keyword based. The use of keywords may be a very effective way to search for images if every image is described exhaustively and the image-searcher has a clear idea of which descriptors to use, given the set of available descriptors. However, keywords that are associated to images can be incomplete and can often be bad descriptors (e.g. an image called *Image01.jpg* does not tell anything about its content; a picture of a boy playing soccer may be named *tiger.jpg* since he was playing '*like a tiger*'). Moreover, the user may have neither a clear description of what she is looking for (other than a picture in her mind), nor the knowledge of all possible good descriptors to choose from given her idea. For all these reasons, the inclusion of image features (color, texture, shape, ..., etc) seems like a reasonable way to improve the image search results.

On the other hand, current research in CBIR (Content Based Image Retrieval) such as *Blobworld* (http://elib .cs.berkeley.edu/photos/blobworld/), *FIDS* (http://www.aa-lab.cs.uu.nl/cbirsurvey/cbir-survey/node16.html), *SIMBA* (http://simb a.informatik.uni-freiburg.de/), *WEBSEEk* (http://persia.ee. colu-mbia.edu:8008/), and *Simplicity*(http://wang.ist.psu.ed u/IM-AGE/), ..., etc, have illustrated the merits of exploiting low level image features for image retrieval. However, on *large scale Web resources*, most CBIR systems' performance may become questionable from a scalability point of view. That is, they have neither been designed to, nor can be expected to index too many images compared to what large scale search engines do. Moreover, they generally do not provide keyword based search functionalities which may be useful in certain search scenarios, depending on the collection at hand, and depending on the user's information need. As can be seen from Table 1, the two methods have the potential to complement each other. If combined together into one search tool, then this tool could tap on the combined strengths of a wealth of well-developed text retrieval techniques and the robustness and scalability of current Search Engine technologies to handle image retrieval. They could also provide a hybrid functionality that enables users to express their information need not only in high level textual format, but also in low-level image feature format, and possibly using a query image as example. A combined tool also promises to alleviate the weaknesses of either type of search working in isolation.

In this paper, we present a seamless combination of the two approaches in an integrated image search system, which can be accessed through: *http://webmining.spd.louisville.edu:8080*.

Table 1: Comparison of Keyword and Content Based Search

Type	Framework	Weakness	Strength
Keyword Based	Search Engines	Information Expression Difficulties	Well-Developed Text Retrieval Methods;Scalability;Easy to Use
Content Based	CBIR	Semantic Gap and Scalability	Ability to express the user's information need using content

2. RELATED WORK

One of the earliest content based image retrieval systems was *QBIC* [12], which presented a user interface that enables users to organize their own queries by choosing a query image or manipulating image features by drawing a sketch or choosing a color from a color wheel. Later, *Blobworld* [29], [7], [5] combined color, texture, and position features of an image to form the feature vectors and used the Expectation-Maximization (EM) algorithm to segment the image into different regions (*blobs*). The system also allowed users to select relevant blobs and corresponding weights for matching. A more recent system, *SIMPLIcity* [33], [18] is based on an Integrated Region Matching(*IRM*) method. Other content-based image retrieval systems include *WALRUS* [22], which extracts the image features using wavelet transforms and sliding windows, then computes the image similarity by analyzing the region matches; *CIRES* [15], which combines structure with color and texture features for analyzing the images, and perceptual grouping for extracting structure information; *C-BIRD* [20], which allows users to search by illumination invariance and

Table 2: Comparison of Different Image Search Systems

System Name	Indexing		Querying		
	textual	content	text	img example	img feature
Google	√		√		
QBIC		√		√	√
BlobWorld		√		√	
SIMPLicity		√		√	
WebSeer	√	√	√		
FIRE	√	√		√	
Webface	√	√		√	
"Show and Tell"	√	√	√	√	√

object modeling capabilities; as well as several other systems [30] [19] [13].

All these CBIR systems are based on image content features such as color, texture, shape, structure, ..., etc. Some systems choose to segment the images, and decompose the whole image into smaller regions to perform matches. However, image segmentation itself is still an unsolved problem. Also, these systems seldom take any textual information or image annotations into consideration. One well known reason for missing this important feature is the notorious *semantic gap* between the image content and the image annotations and the fact that different people may have different annotations for the same image due to their different perspectives and domain knowledge. However, this should not lead us to neglect the wealth of information contained in people's description of images using *words*. Another reason why text-based information is often overlooked, may be that most CBIR systems' application domains and image sources tend to be narrow, and they generally do not consider the entire breadth of the World Wide Web as a source of their searchable images. When dealing with images from the World Wide Web, where the image source is not narrow, human tags or annotations can play an important role in differentiating between different images.

Some image retrieval systems have recently started considering their image source from Web collections, and also considering textual information. For example, *WebSeer* [31] combines textual and image content information, although the image content part deals largely with image *metadata* that are contained in the header of an image file (such as image size, creation date etc.). Recently, *FIRE* [11], which was developed for medical image retrieval, also combined textual and image content information. Finally, *Webfaces* [1] was developed as a face detection engine by using both the image textual and content features. However the input query can only be in the form of an image and not text. These and other similar systems [8] [6] combine textual and content information for image retrieval, but the way they use textual information is mostly for building internal relationships or models between a set of training images and corresponding textual information (typically the names of the subjects whose faces are in the images) by using machine learning, classification, clustering, or Latent Semantic Indexing. The textual information in these systems remains hidden to users; that is, users are not allowed to query the system using these textual keywords. Hence, they share limitations that are similar to the above mentioned CBIR systems when it comes to the user interface. Users in these systems are only allowed to query by submitting an example query image [16], or by progressively browsing a set of images [35]. For a comparison, see Table 2.

3. CONTRIBUTIONS

Even though the *indexing* component cannot be de-emphasized when assessing the capabilities of an image retrieval system, the importance of the *querying* component has often been neglected from a quality perspective, and has instead been only emphasized from a user-friendliness or flexibility point of view, along the line of interest in Human Computer Interaction. However, this attitude overlooks the important role that a user's Information Need, and the way that it can be expressed, plays in most real-life search scenarios. Regardless of the effort spent in modeling and indexing information, if the user cannot accurately express their information need, then all these spent efforts risk being in vain. Humans often harness the full potential of their senses (including *visual* senses) to process information, and typically *communicate* this information through *language* or *words*. Hence, *both linguistic* (*text*) and *visual* (*image content*) are inseparable and crucial components for expressing a user's *information need*, and thus for formulating *queries*.

From Table 2, we can see that the earliest systems focused either on the textual part or the image content part, but always exclusively one or the other. For other system that combine text and content together, the final *interface* for a user's query turns out to be either contented based or keyword based, where one feature is exclusive of the other. On the other hand, the proposed system, that we named *"Show and Tell"*, can handle the user's query both by using *text/keywords*, as in current commercial image search engines such as *Google* and *Yahoo*; as well as by using an *example image* or using a description of *image content features*, as in most CBIR systems. Most importantly, the proposed system allows users to input *Boolean queries* that *combine* all these different features together to express a myriad of possible conditions. For example, users can search for images using the combined *word* and image *feature* query "*roses imgcolor:red*" to find pictures of *red roses*, or they can issue the query "*roses -imgcolor:white*" to search for pictures of *roses that do not contain the white color*.

Overall, the following list summarizes our contributions and the functionalities of the proposed tool.

1. We build an image search engine based on modifying the *Lucene* based open source search engine "nutch". In this way, we extend nutch's powerful capability of indexing and searching to image retrieval.

2. We encode the *low level image features* (limited to using a color histogram in this paper) into *text* so that nutch can index and search them. We then combine the color features with textual annotations (or tags) of the image to yield a seamlessly integrated and indexable representations of the images in a text-like domain. Hence, the tool can be used to deal with World Wide Web image collections. This approach can improve current commercial image search engines so that they can handle the content part of images. Also it compensates for the inefficiency of some content-based image retrieval systems by exploiting a full-fledged crawling, indexing and searching scheme.

3. We not only provide the query *by keyword* capability that is used in current commercial image search engines such as *Google*, but also provide the *query by example image* compatibility that is widely used in current CBIR systems. And most importantly, we provide the *query by image feature* (currently illustrated by color features) capability, which is seldom addressed by both of the above two types of image retrieval systems.

4. Our real-time clustering capability can help the user locate the desired images more quickly. By real-time clustering, we mean that the clustering process is performed live (if the user chooses to), after obtaining the search results, in a similar manner to the text-based search engine *Vivissimo*.

5. We have conducted our experiments for different distance measures over 5,000 images from the Corel database (to provide a controlled experimental environment), in order to study the precision and recall of the results.

A demo of the developed tool can be visited through the URL: *http://webmining.spd.louisville.edu:8080*. We organize the paper as follows. In Sections 4 and 5, we introduce our main method of indexing and querying images. Section 6 describes our implementation, while Section 7 gives the experiments and evaluations. Finally Section 8 concludes the paper.

4. INDEXING IMAGES

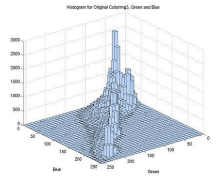
Similar to many other image retrieval systems, we decided to start with color features for image representation because Human eyes are most sensitive to this feature. Also, the color histogram is easy to compute, and tends to be robust against image transformations such as shift and rotation. Even though we currently index only color features, it should be clear that this framework could be extended to other features, such as texture and shape, which are left for future work.

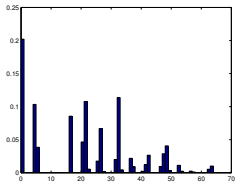
4.1 Colorwords

Current indexing and retrieval techniques for text documents, such as Web pages, pdf documents, etc, are nearly full-fledged. Most of them use an inverted list for indexing. That is, they decompose the documents into keywords, and build the index upon such keywords. The general ranking of the returned results is generally based on the *TF-IDF* method, in addition to Web citation or *link analysis*, and web page analysis. When a user submits a query containing several keywords, the search system would return the documents that contain these keywords.
Inspired by this framework, we decided to encode the image features into *text-like keywords*, and therefore tap on the strengths of the *storing*, *indexing* and *retrieval* techniques used for processing text documents. We then build the index based on the keywords encoded from the image features. Although we currently use only the color histogram for encoding features, the principle can be extended to include other image content features. We name the encoded color feature keywords as "*colorwords*". Just like a web page or pdf document may contain many keywords, an image may contain many colorwords.
As mentioned above, the ranking of resulting documents is usually based on the TF-IDF method. That is, when a user searches for a keyword, if one document contains many such keywords (their term frequency is high), this document would be ranked higher in the returned list for that keyword. In contrast, if the keyword is contained in too many documents, (such as "this", "that", ..., etc), then its IDF value will be low, and it will not play the role of a good descriptor for a document. Hence, its effect on the ranking of the documents will be lower. Similarly, we can transplant such document-based concepts into the image and colorword domain. If an image contains a high frequency of a specific colorword, and a user searches for that specific colorword, then the ranking for that image would be higher. On the other hand, if a colorword, such as one corresponding to background color, is contained in too many images, then it will not be a good descriptor for searching images. Hence, its effect on ranking the resulting images would be lower than other less frequent colorwords
Intuitively, when the user searches for color "*red*", the retrieval system will rank the images that contain more "red" components higher since their "*TF*" value is higher. And if the user issues a boolean query "*violet + black*", the images that contain a given level of "*violet*" would generally be ranked higher than the images that contain a comparable level of "*black*" since generally, more pictures tend to contain "*black*" than "*violet*". hence, the *IDF* value of the colorword "*black*" is expected to be lower than that of "*violet*".

4.2 Color Histogram And their Colorword Representation

The *color histogram* has widely been used in most current CBIR systems since it has proved to be a dominant feature for identifying an image. Although it cannot tell the specific location of an object in an image, it does summarize the general color components of an image and it has the merits of being robust to noise and image transformations such as rotation and shift. Therefore, in our implementation, we use the global color histogram to represent the image content features.
The 3-D (in *RGB* or *Red-Blue-Green* space) color histogram is hard to visualize, but the 2-D color histogram can be visualized as in Figure 1.





ColorWord	Proportion(term frequency)
Colorword_1	0.202
Colorword_33	0.114
Colorword_22	0.108
Colorword_5	0.103
Colorword_17	0.085
Colorword_21	0.066
Colorword_27	0.047
Coloword_49	0.041

Figure 1: From image to *colorwords*: clockwise from top left: original image, color histogram 2-D projection, *colorwords* and their corresponding frequencies (*TF*), and finally color histogram on 64 bins

One important question concerns the number of bins to use. If too many bins are used, the computation will become a burden, while if too few bins are used, some information will be lost. In our implementation, we used 64 (4x4x4) bins with 4 values for each of Red, Blue, and Green color components.
The transformation process is shown in Figure 1. For 64 bins, each bin corresponds to a color index number, which is the centroid of the color indices contained in that bin. After counting the occurrence of the image pixels that fall into each bin, we obtain the 3-D color histogram. We then normalize the histogram so that large pictures and small pictures can be comparable from a color perspective, and we encode these 64 indices into *text* (i.e. *words*). When indexing an image using nutch, we do not use all the 64 bin values of the image since according to our analysis, for an image, only a few bin values of the total 64 values tend to stand out, while most other values are very small in comparison. For our implementation, we used the 8 most important values (colors) of the 64 bin color histogram to represent an image.
When building the index using nutch, to exploit the *TF-IDF* matching mechanism of the search engine, we map the 8 textual index values to integers, so that the index with the smallest value is mapped to 1. For the larger values (that are n times as frequent as the lowest value), we just repeat their colorterms n times when building the index. This trick was done because search engines index only keywords that are *countable*.

4.3 Combining Image Colorwords and Textual Information for Indexing

The image *textual* information or annotations are extracted from World Wide Web image collections by parsing the following components:

1. *Image URL*: Generally, the image URL can contain important information about the annotations of an image. Thus, it may contain the image category and image name information, which provide good image annotations. We exploit *Nutch*'s powerful functionality to parse and tokenize the image URL and extract these image annotations.

2. *Image Anchor Text*: The Anchor text is another important source for getting the annotations, since it provides a description of the link to the image from inside a Web page.

Certainly these sources may not be perfect. For example, an image named by a number cannot yield a useful annotation, though its anchor might do so. Also, the above sources may not be sufficient to extract all the image annotations. We left for future work, the extraction of annotations from the HTML file that contains the image. This may include "*ALT*" tags, more text

Table 3: Image Color Palette

Red	Green	Blue	Cyan	Magenta	Yellow
Brown	Orange	Violet	Gray	Black	White

located in the vicinity of the anchor text, as well as other tags that may be available when images are embedded in XML documents.

Figure 4.3 illustrates how the information obtained after parsing an image contents, i.e. its *colorwords*, and parsing the image *textual* annotations from the parent HTML file, is fed to *Nutch* for building an inverted searchable index.

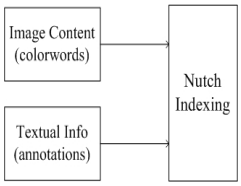


Figure 2: Indexing Images

5. QUERYING THE SEARCHABLE INDEX

Some users may not be inclined to search for one specific colorword. Hence, some user friendly search capabilities need to be provided, as explained below.

5.1 Query by Image Color-Image Color palette

After the colorwords have been extracted from the image and indexed, we need to provide a search function for them. For this end, we need to correlate the colorwords with human beings' linguistic color descriptions such as *red, blue, brown*, ..., etc. The image color description of human beings are individual dependant. For example, some users might describe gray as light black, while others might describe certain dark shades of pink as almost red. For these reasons, we use an *image color palette*[1]. In our color palette, we use 12 colors as shown in Table 3. Notice that the color definitions are not categorical or clear-cut. Each of the above colors forms a group of several similar colorwords and each colorword may belong to multiple groups. For example, for the color red, we not only assign the color with RGB value FF0000 (in our 64bins, it is contained in the bin with RGB value E00000), but also assign the color with RGB value equal to A02020 and 602020, which are colors between brown and dark red. To formulate a query, we use a Boolean logic "OR" so that all the pictures that contain a color that is close to red are retrieved. Here the traditional information retrieval technique *TF-IDF* plays a reincarnated role of matching and ranking according to *textual* and *image content* (except that *image content* has been mapped to *feature words*). Recall that during the indexing process, we repeated a colorword n times of if this colorword occured n times in the image's 3-D color histogram. So if one colorword occurs more frequently, its *TF* value will be large, and therefore, the image will be ranked higher than other images. For example, if a user submits the query "*roses imgcolor:red*", the results that contain the keyword "*roses*" and contain more of the *color red* will be ranked higher than the ones containing less of the red color component.

5.2 Query by Image Example

From Table 2, we can see the Query by image example option is provided by most CBIR systems. In this case, the system should return results that are ranked by similarity with the query images. Besides the widely used Euclidean distance and Quadratic distance, some other distance measure like Cosine angle distance [23], Earth Mover's Distance, [28] and other distance measures have been explored in several systems [27][24] [32] [34] [21].

For real-time image retrieval, there is a tradeoff between retrieval speed and similarity measure accuracy. For example, when using color histograms, if too few bins are chosen, the image representation will be coarse and may lose significant color information; while if too many bins are chosen, the high dimensionality will impose a big computational burden. For this reason, some methods use adaptive binning [17] and dominant colors or representative color [25] [10] to represent images efficiently while reducing the dimensionality.

Putting our concern in the effectiveness and efficiency of the image search process, we tested three similarity measures: Cosine Angle Distance, Euclidean Distance, and Quadratic Distance.

Since colorwords are used to encode the image color content, they will be exploited to calculate the image similarity. The cosine similarity, which is commonly used to compare term vectors, is given by

$$SIM(Q, D_i) = \frac{\sum_i W_{Q,j} W_{i,j}}{\sqrt{\sum_j W_{Q,j}^2}\sqrt{\sum_i W_{i,j}^2}} \quad (1)$$

where Q is a query image, D is a document (image) relevant to Q and W are weights. The normalized term frequency is used for the weights as follows

$$W_{ij} = TF_{ij}/TF_{i,max}$$

where TF_{ij} is the frequency of term j in Document/image i, and $TF_{i,max}$ is the maximal TF frequency among all terms in Document/image i. During the indexing phase, we encode the 8 most important color components into colorwords. However, when we used the boolean logic "AND" between the colorwords, only one image (the query image) was returned. In this case, it will not be comparable with other results. Hence, we reduced the resolution to 5, which can still give a meaningful number of results for evaluation.

The Cosine similarity tends to benefit from a low computational cost without sacrificing much accuracy. This is especially true because most images contain only several important color components. We have also explored the Euclidean and Quadratic distance measures from the point of view of image ranking and computational efficiency. The Euclidean distance is given by

$$d_{Q,D} = \sqrt{\sum_{k=1}^{n}(Q_k - D_k)^2} \quad (2)$$

Note that if all vectors are normalized, then the ranking result of the Cosine Angle distance and Euclidean distance can easily be shown to be identical. See [23] for further information.

For Quadratic Distance is given by

$$d_A(D, Q) = \sqrt{(D-Q)^T A (D-Q)} \quad (3)$$

Where A is the cross-bin similarity matrix $A = [a_{i,j}]$, and $a_{i,j}$ denotes the rank similarity $|i - j|$ between bin i and bin j.

5.3 Realtime Clustering

Clustering was used in [36],[2], [3] and [9] for different purposes. We use it to obtain a more user friendly interface. The realtime clustering is helpful to quickly locate the images that a user is seeking. For example, if a user inputs a keyword to the search engine and enables the clustering option, he or she will see a few representative images in each cluster (currently, we display all the images contained in each cluster). The user can then click on the image in any of the clusters to refine their query based on that image as example. This will play the role of *zooming* in search. In real applications, it may not be feasible to show all the images retrieved in the first page, but the user may want to get a whole picture of all the results to avoid resorting to the next page blindly.

[1]http://webmining.spd.louisville.edu:8090/images/color palette.html

We use K-means to cluster the image color features. Choosing the number of clusters is beyond the scope of this paper. Hence we chose $k = \sqrt{N}$, for N result images. We use the extracted image tags as an easy way to compute the purity of clusters.

6. IMPLEMENTATION

6.1 A Brief Overview of Nutch

Nutch[2] is an open source search engine application. It uses *Lucene* for the search and indexing component, and a powerful fetcher (crawler robot). Nutch has a highly modular architecture allowing developers to create plugins for the following activities: media-type parsing, data retrieval, querying, and clustering. In June 2003 there was a successful 100 million page demo system. See [4], [14], and [26] for further information.

6.2 Image Sources and Crawling

Nutch has its own robot for crawling the World Wide Web. Currently it can parse various file formats such as html, pdf, rtf, ..., etc, but it does not have a plugin to parse images. Hence, we wrote our own image parsing plugin, *Jpeg-parser*, that can parse .jpg images. During the crawling process, Nutch's crawler fetches and parses web pages from the World Wide Web, and when the specific file types are found, it invokes the corresponding parsers to handle this file. To get the textual information of the images or the images tags, we need to find a method that matches the image annotations to the image. However, the annotations can occur anywhere within an HTML file. For example, the image annotations can be in the image file name, the image URL, the "ALT" text, the anchor text, etc.

First, we started by using the image file name and the URL as image annotations. But for some web sites, the image file name are just numbers and the image URL has no relation with the image content. Therefore, we used only several well formed image repositories: *Washington* database[3], *Bigfoto* http://www.bigfoto.com/, and the SIMPLICity[4] annotated database that has been stored on our server to be crawled on a private port[5] for experiment.

During the crawling process, the HTML parser is invoked to extract the image links, and the jpeg parser will then be used to handle the image links. After crawling, the indexing part is invoked to build an index so that all crawled data becomes searchable. During the indexing period, we only build the index for the jpeg images and skip other file formats.

6.3 Modifying Nutch's Settings and Code for Image Crawling and Indexing

By default, nutch cannot parse images, in order to handle the image indexing and querying, we had to customize it for dealing with images. This includes modifying nutch's configuration file, Indexer, Searcher, and Clustering parts for image searching purposes. One of the most desirable features of nutch is the ability to expand its functionality by writing plugins that are not invasive to its inner indexer and searcher. Hence, we developed the following plugins.

1. *Image Parsing Plugin -JpegParser.* This is to handle jpeg images in the crawling and indexing phase.

2. *Online Clustering Plugin -clustering-imgfeatures.* We added this module to use low level image features for grouping images.

3. *Image Query Plugin -query-imgURL.* In the same spirit as the query by site (*site: keyword*) and query by URL (*URL: keyword*) functionalities, we developed the query by image URL (*imgURL: keyword*) plugin. And it is for this functionality that the customized sorting plays a role.

[2]http://lucene.apache.org/nutch/
[3]http://www.cs.washington.edu/research/imagedatabase/groundtruth/
[4]http://wang.ist.psu.edu
[5]http://webmining.spd.louisville.edu:8085/

Table 4: Query Syntax and Examples

Query Type	Examples
Keyword	*roses*
Image Color	*imgcolor:red*
Image Example	*imgURL:http://xxx.jpg*
+(-)Keyword +(-)Image Color	*roses -imgcolor:red*
+(-)Keyword +Image Example	*roses imgURL:http://xxx.jpg*
+(-)Image Color +Image Example	*imgcolor:red imgURL:http://xxx.jpg*

Note: by default, Boolean "AND" is used

Table 5: Categories and Subcategories of Tested Images

Category	Subcategories
Food	barbecue;vegetable;dining;fruit;seeds
Flower	roses;orchids;cactus;flowerbeds;msc
Vehicle	British car;buses;trains;Rare car;transport
Animals	tigers;elephants;horses;antelope;lion
People	youth;women;men;tribal;model;
City	Bali;London;Hongkong;Hawii;Toronto
Country	Italy;Africa;Canada;Kenya;Egypt
Art	Bonsai;Dinosaur art;Bird art;craft;painting;
Sports	polo;surfing;golf;skiing;msc
Scene	mountain;desert;beach;dawn.dusk;msc

4. *Image Feature Query Plugin -query-imgcolor.* Based on this plugin, users are allowed to query by image colors.

6.4 User Interface And Query Functionalities

Table 4 lists he syntax and examples of six different querying modes implemented so far, including querying by text keywords, image features (currently color), and image example, as well as Boolean combinations thereof. For some snapshots of query examples, see the attached figures.

7. EVALUATION AND EXPERIMENTAL RESULTS

In our evaluation, we used 10 categories, (see Table 5), from the COREL image database, where each category contains 5 semantically coherent sub-categories, Altogether, there are 5000 images in total for testing. Our testing is conducted by combining the keyword (category name) and an image from a subcategory as query. We count the first 10, 20, ...100 result set to see how many result images fall into the same subcategory as the query image and transform this as precision. We chose one image from each subcategory and altogether got 50 queries. We averaged the 50 result sets to avoid random results. Figure 3 shows an example test result distribution for animals. For the animals case the precision is for the first 100 result set and is calculated as follows:

$$p'_{average} = (p'_1 + p'_2 + p'_3 + p'_4 + p'_5)/5 = 1$$
$$= (0.46 + 0.66 + 0.33 + 0.32 + 0.56)/5 = 0.466 \quad (4)$$

For 50 queries we calculated the average precision by summing all precisions divided by 50. Figure 4 shows the experiment results for three different distance measures. We ran this test on our a linux system, with Intel(R) Xeon(TM) CPU 2.80GHz, 2cpu, with 2G memory. The precision for the Euclidean Distance measure is slightly better than Cosine Angle Distance. The computationally intensive Quadratic Distance measure doesn't exhibit any benefit as expected, although it considers the cross-bin similarity. From a computational speed perspective, Cosine angle measure wins without doubt with an average speed of 0.236s, compared to Euclidean distance (1.837s) and Quadratic Distance(7.104s).

8. CONCLUSIONS AND FUTURE WORK

For image search, keywords can play a filtering role, while image content plays a semantic role for getting better visual results.

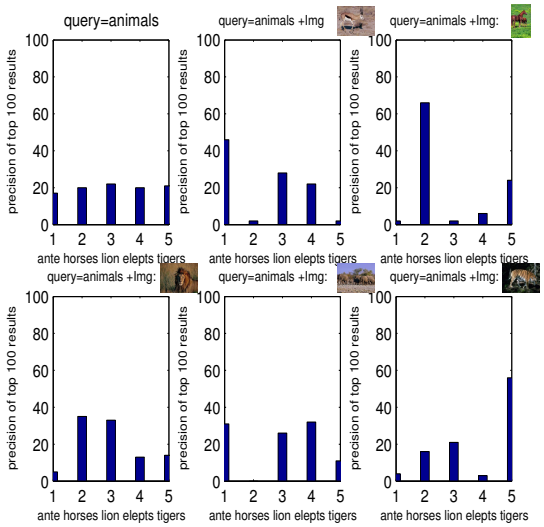


Figure 3: Example Query Result Distribution

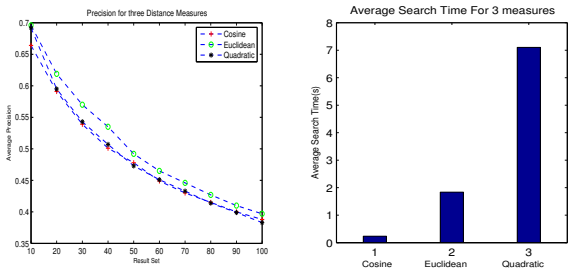


Figure 4: Experiment Result

Using only one of them in a query could lead to false positives. For example, using image *content* query *alone*, a "*rose*" image may be deemed very similar to an image of "*a woman wearing a red hat*", although the two images are distinct. Another example is when a user searches for an image of a tiger by using the *keyword* "*tiger*"as query, and getting as result the image of a man named "*tiger*" such as "*Tiger Woods*". However, the *combination* of the image *keyword* and image *content together* in the same query, would reduce the occurrence of the above false positives.

We have described our development of a search tool benefiting from an integration of text and image content in querying and indexing. This framework promises to inherit the scalability and performance of powerful text search engines. For future work, we consider crawling the world wide web images to get annotations and extending our method to include image texture, shape and other information to improve the performance.

9. REFERENCES

[1] R. Baeza-Yates, J. R. del Solar, R. Verschae, C. Castillo, and C. Hurtado. Content-based image retrieval and characterization on specific web collections. *Conference on Image and Video Retrieval (CIVR), Springer LNCS*, pages 189–198, 2004.

[2] K. Barnard, P. Duygulu, and D. Forsyth. Clustering art. *Computer Vision and Pattern Recognition*, pages 435–439, 2001.

[3] K. Barnard and D. Forsyth. Learning the semantics of words and pictures. *International Conference on Computer Vision*, 2:408–415, 2001.

[4] M. Cafarella and D. Cutting. Building nutch: Open source search. *ACM Queue*, 2(5), April 2004.

[5] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Blobworld: Image segmentation using expectation-maximization and its application to image querying. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(8), August 2002.

[6] C. Carson and V. E. Ogle. Storage and retrieval of feature data for a very large online image collection. *IEEE Computer Society Bulletin of the Technical Committee on Data Engineering*, 19(4), December 1996.

[7] C. Carson, M. Thomas, S. Belongie, J. Hellerstein, and J. Malik. Blobworld: A system for region-based image indexing and retrieval. *Proc. Third International Conf. Visual Information Systems*, pages 509–516, 1999.

[8] M. L. Cascia, S. Sethi, and S. Sclaroff. Combining textual and visual cues for content-based image retrieval on the world wide web. *Proceedings of the IEEE Workshop on Content - Based Access of Image and Video Libraries(CBAIVL)*, 1998.

[9] Y. Chen, J. Z. Wang, and R. Krovetz. An unsupervised learning approach to content-based image retrieval. *Proc. IEEE International Symposium on Signal Processing and its Applications*, pages 197–200, July 2003.

[10] Y. Deng, B. S. Manjunath, a. M. C. Kenney, and H.Shin. An efficient color representation for image retrieval. *IEEE Transactions on Image Processing*, 10(1):140–147, Jan 2001.

[11] T. Deselaers, T. Weyand, D. Keysers, W. Macherey, and H. Ney. Fire in imageclef 2005: Combining content-based image retrieval with textual information retrieval. *Working Notes of the CLEF Workshop, Vienna, Austria*, September 2005.

[12] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, and et al. Query by image and video content: the qbic system. *IEEE computer*, 28(9):23–32, Sept 1995.

[13] Y. Gong. Advancing content-based image retrieval by exploiting image color and region features. *Multimedia Systems*, 7(6), 1999.

[14] E. H. O. GOSPODNETIC. *Lucene in Action*. Manning Publications Co., Greenwich, CT, 2005.

[15] Q. Iqbal and J. Aggarwal. Cires: A system for content-based retrieval in digital image libraries. pages 205–210. 7 International Conference on Control Automation, Robotics and Vision (ICARCV), December 2002.

[16] V. Kovalev and S. Volmer. Color co-occurrence descriptors for querying-by-example. *Proc. of the 5th Int'l Conf. on Multimedia Modeling*, pages 32–38, October 1998.

[17] W. K. Leow and R. Li. Adaptive binning and dissimilarity measure for image retrieval and classification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2:234–239, 2001.

[18] J. Li and J. Wang. Automatic linguistic indexing of pictures by a statistical modeling approach. *IEEE Trans. on PAMI*, 25(9), 2003.

[19] X. Li, S.-C. Chen, M.-L. Shyu, and B. Furht. An effective content-based visual image retrieval system. *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, 2002.

[20] Z.-N. Li, O. R. Zaiane, and B. Yan. C-BIRD: Content-based image retrieval from digital libraries using illumination invariance and recognition kernel. In *DEXA Workshop*, pages 361–366, 1998.

[21] Q. Lv, M. Charikar, and K. Li. Image similarity search with compact data structures. *In Proceedings of the 13th ACM Conference on Information and Knowledge Management (CIKM'04), Washington,D.C.*, November 2004.

[22] A. Natsev, R. Rastogi, and K. Shim. WALRUS: a similarity retrieval algorithm for image databases. SIGMOD, Philadelphia, PA, 1999.

[23] G. Qian, S. Sural, Y. Gu, and S. Pramanik. Similarity

between euclidean and cosine angle distance for nearest neighbor queries. *SAC'04*, March 2004.

[24] S. Ravela and R. Manmatha. On computing global similarity in images. *Proceedings of the 4th IEEE Workshop on Applications of Computer Vision (WACV'98)*, 1998.

[25] J. Ren, Y. Shen, and L. Guo. A novel image retrieval based on representative colors. *Image and Vision Computing New Zealand*, 2003.

[26] K. S. A. R. Rohit Khare, Doug Cutting. Nutch: A flexible and scalable open-source web search engine. In *CommerceNet Labs Technical Report 04-04*, November 2004.

[27] Y. Rubner, J. Puzicha, C. Tomasi, and J. M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. *Computer Vision and Image Understanding*, 84(1):25–43, October 2001.

[28] Y. Rubner, C. Tomasi, and L. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[29] S.Belongie, C.Carson, H.Greenspan, and J.Malik. Color and texture-based image segmentation using em and its application to content-based image retrieval. *IEEE ICCV*, pages 675–682, Jan 1998.

[30] T. K. SHIH, J.-Y. HUANG, C.-S. WANG, J. C. HUNG, and C.-H. KAO. An intelligent content-based image retrieval system based on color , shape and spatial relations. *Proc. Natl. Sci. Counc. ROC(A)*, 25(4), 2001.

[31] M. J. Swain, C. Frankel, and V. Athitsos. Webseer: An image search engine for the world wide web. *IEEE Computer Vision and Pattern Recognition Conference*, 1997.

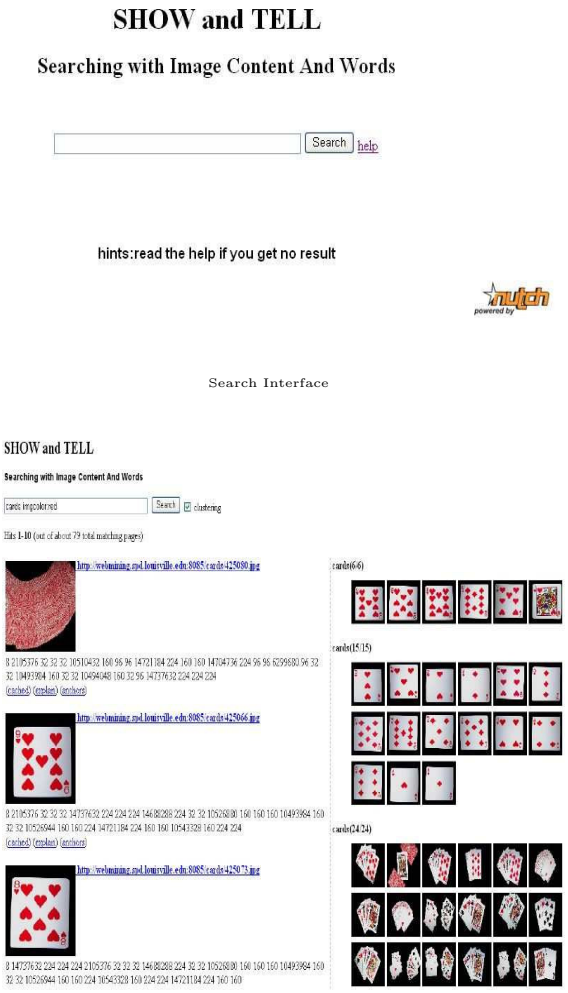
[32] K. Vu, K. A. Hua, and J. Oh. A noise-free similarity model for image retrieval systems. *Proc. of IS&T/SPIE conference on Storage and Retrieval for Media Databases*, pages 1–11, Jan 2001.

[33] J. Z. Wang, J. Li, and G. Wiederhold. Simplicity: Semantics-sensitive integrated matching for picture libraries. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 23(9), Sept 2001.

[34] W. Xiaoling and X. Kanglin. Application of the fuzzy logic in content-based image retrieval. *JCS&T*, 5, April 2005.

[35] C. C. Yang. Content based image retrieval: A comparison between query by example and image browsing map approaches. *Journal of Information Science*, 30(3):257–270, 2004.

[36] R. Zhang and Z. M. Zhang. A clustering based approach to efficient image retrieval. *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'02)*, 2002.



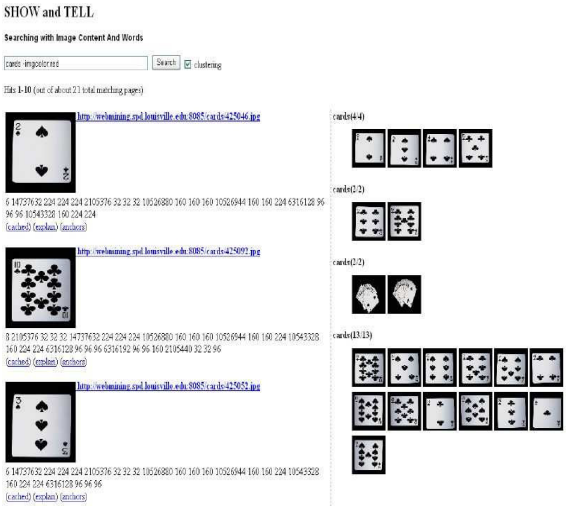
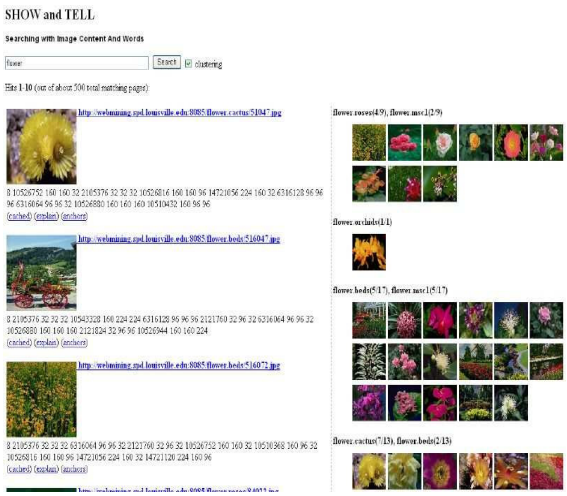
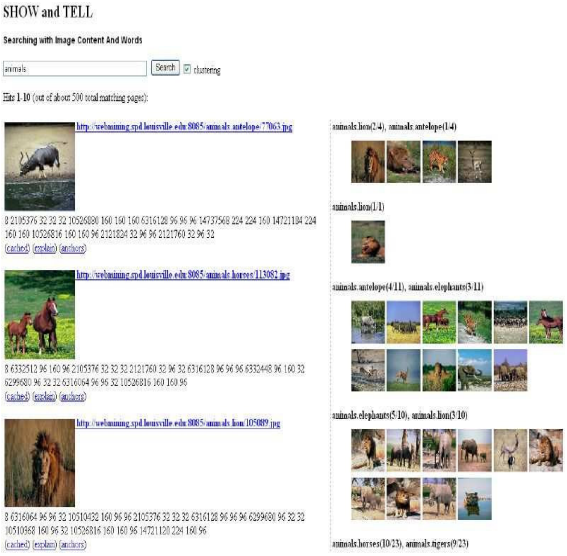


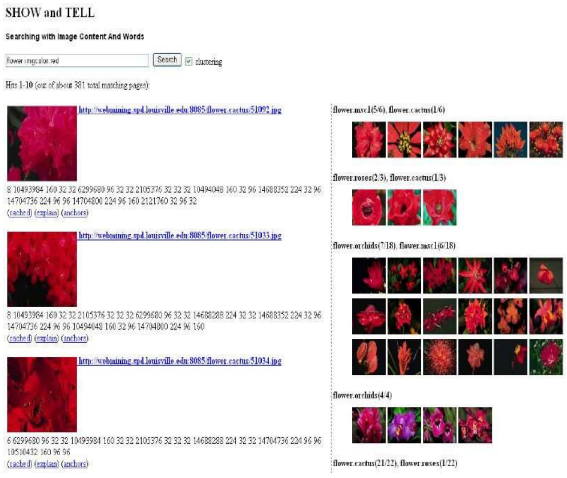
Figure 5: Interface And Query Example 1



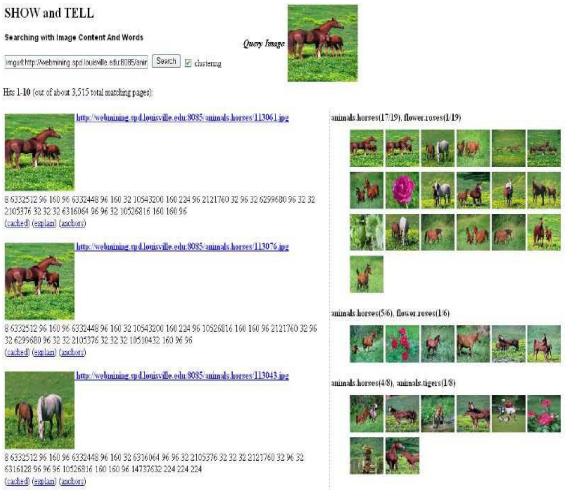
Query = *flower*



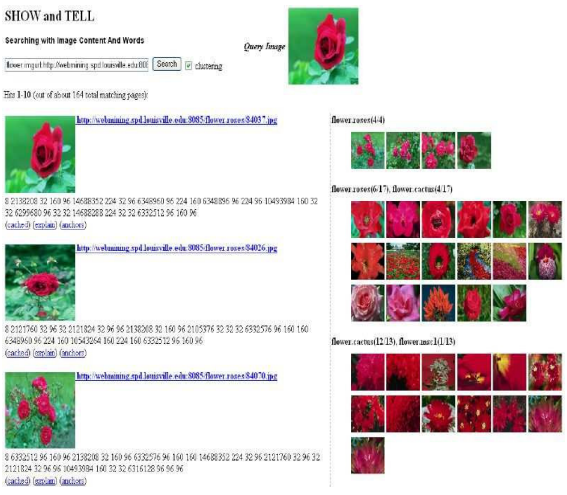
Query = *animals*

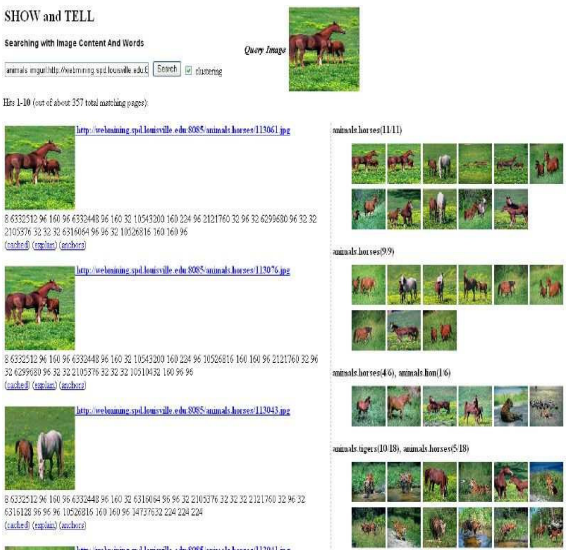


Query = *flower imgcolor:red*



Query = *imgURL:http://webmining.spd.louisville.edu:8085/animals.horses/113061.jpg*





Standards for Open Source Information Retrieval

Wray Buntine
Helsinki Institute of
Information Technology
University of Helsinki, Finland
buntine@hiit.fi

Michael P. Taylor
Index Data Aps.
London, England
mike@indexdata.dk

Francois Lagunas
Exalead
Paris, France
lagunas@exalead.com

ABSTRACT

Standards are important because they make a field more open to small and medium businesses and to academic players. We review a number of standards that apply to information retrieval and web search, and discuss the role that they play. We also discuss some areas where there is potential for the development of standards, where for instance information retrieval would benefit, and where standards development appears feasible.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

1. INTRODUCTION

The various standards for communications on the web, such as HTML and HTTP, SMTP, FTP, MIME, URI's and soforth, are the key infrastructure that made the phenomenal commercial growth of the web possible (arguably many other factors exist including including hardware and telecommunications, academic incubation, etc.). These standards allowed small, medium and large businesses to join in the development of products and services. A similar phenomenon occurred before this with IBM PCs and their standard hardware architecture. The web has proven to be a true revolution in business whose effects are still being seen (for instance web advertising recently overtook some forms of print in total expenditure). The importance of standards in creating an open playing field should not be underestimated.

With this motivation in mind, in this paper we review some of the major standards available to the information retrieval (IR) and web search communities. This serves two purposes: first, to survey the field and understand the current offerings; and second, to form a basis for discussion on the penetration of these standards and of new areas where standards might well be used.

Our approach to standards w.r.t. IR are presented in the second section. Then, in the third section, a number of fairly well established standards are presented and discussed. Information extraction is discussed separately because we believe it is important for the future of IR to extend its semantic capabilities, yet no standards currently exist. Finally, some more speculative proposals are considered.

2. ON STANDARDS

This section outlines some aspects of standards that are best clarified before covering the standards themselves.

2.1 What is a standard?

The standards we include in this discussion are taken from those listed at our website, `OpenSourceSearch.ORG`, that have been pointed out to us over time, or have been used by our own group. Notice that we exclude from the discussion standards in the area of traditional web activities, web services, web commerce and semantic web. The former are rather orthogonal to the issues of IR, and may be combined as needed. The latter, the semantic web, will play a growing role in IR, a role still being understood while semantic web standards themselves are in flux.

Moreover, we use the term standards rather loosely. Officially, standards should have been approved by some national or international body such as ISO, NIST, IEEE, etc. We also use the term to include proposed standards, *de facto* standards, protocols that have been published, perhaps by a commercial organisation, and have become in the colloquial sense, "standard."

2.2 Standards not covered

Finally, a special mention needs to be made of XML even though it falls in the category of a general and somewhat orthogonal standard. Embedding annotation (for instance, named entities) and document structure (for instance sections and subtitles) is generally done with XML. Moreover, it is the basis for semantic web standards such as the Resource Description Framework (RDF). When XML is used, languages such as XSL and XQuery become available. While XML is not an efficient format for communication, we look forward to advances from the Efficient XML Interchange (EXI) initiative. Thus, while XML and tools are useful and form an underlying technology, they are not critical standards to affect the direction of open source information retrieval.

Another standard here that should be mentioned is JXTA[1]: "a set of open protocols that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner." While it is a well known P2P standard, it is not generally used in distributed or P2P IR work because it is not well targeted for it.

2.3 A framework

In discussing standards, we will use a common summarising framework based around the important issues:

[1] `http://www.jxta.org/`

Target: What is the intended target for the standard. i.e., which group of developers or what functions are being highlighted. Described with the target should be a compelling reason as to why the standard should take on.

Advantages: What key advantages will the adoption of the standard offer.

Barriers: What are the potential barriers to the standard being adopted.

We were considering trying to present a systems or architectural framework for search to explain the various standards and their roles within information retrieval and search. This, however, proved elusive. Suffice it to say, that all standards resolve around an interface between key actors in a search system. Sometimes those actors represent separate businesses or users, and sometimes they represent separate software systems or components.

3. EXISTING STANDARDS

In this section, actual standards and current or emerging *de facto* standards are presented.

3.1 Query and retrieval: CQL and SRU

In the mid-to-late 1990s, the HTTP protocol that underlies the World Wide Web began to be used for more complex operations. Various digital collections were made available for searching by means of web interfaces, including library catalogues, museum collections, staff directories, aggregated article abstracts, and of course databases of web-pages, such as Yahoo, AltaVista and latterly Google. Most of these search UIs worked by using HTML forms to submit an HTTP GET request to a server – in other words, to generate a URL that contained the query together with auxiliary information such as the number of records to retrieve, whether to restrict results to those in a particular language, etc. Despite the growing use of AJAX techniques, the simple HTTP GET URL is still by far the most widely used technique for searching on the web.

In the absence of a standard for encoding rich queries into HTTP URLs, each of these services created their own conventions: for example, when searching for English-language PDF documents that contain the word "dinosaur", Yahoo uses the "va" parameter for the keyword, "vl" for the language (with the value "lang_en") and "vf" for the file format. Meanwhile, AltaVista uses "aqa" for the keyword, "kls" which has the boolean value "1" for English-language results only, and "filetype" for the format. Google is different again, using "as_q", "hl" (this time with the value "en") and "as_filetype". This inconsistency has not been an issue for service providers with no ambitions beyond providing a human-facing Web interface to their search engines, but makes it impossible to write a generic searching client that works across many different engines.

In response to this need, several standardisation processes have proposed wildly different solutions, which vary primarily in how they trade power against simplicity. At one end are the SOAP-based Web Services, which potentially achieve impressive results, but are hobbled by the complexity of the protocol and by numerous different implementations that do not properly interoperate. At the other end of the spectrum is OpenSearch, which has as its primary goal a low barrier to implementation, but which suffers from a correspondingly low level of precision in its queries. Perhaps the best trade-off between these extremes is SRU (Search/Retrieve via URL[2]), a candidate standard sponsored by the Library of Congress and developed jointly by an informal consortium of both public and private organisations from the commercial and academic sectors.

SRU was developed by librarians and engineers with many years' practical experience of the older information retrieval standard ANSI/NISO Z39.50. Thus it benefits from decades of experience of how to create specifications that facilitate semantic interoperability as well as the syntactic interoperability addressed by the other initiatives. The principal lesson of Z39.50 has been that semantic interoperability is both much more difficult and ultimately much more important than syntactic: that the same query can be broadcast to a hundred services is of little value if they interpret it in a hundred different ways.

Thus, from the beginning, SRU has been designed to enable queries to express precise semantics, using the related candidate standard CQL (Common Query Language[3]). CQL provides the means for individual query terms to be related to particular indexes (e.g. "author=farlow and subject=dinosaurs"), which allows a simple interface to semantically tagged content. For instance, a CQL search interface to MedLine can be configured to allow search fields such as Gene, Protein and Species. CQL indexes are taken from *context sets*, analogous to XML namespaces, an arrangement that allows domain specialists to create sets of indexes appropriate for searching in their domain, and provides the basis for simple semantic-based search. CQL context sets can also contain relational modifiers, boolean modifiers, date comparison, and refinement for sorting.

Although SRU and CQL capture most of the power of Z39.50, and add much that is new, their simplicity presents a very low barrier to implementation, and many free toolkits are available to facilitate the development of both clients and servers[4]. SRU has been adopted by the NISO Metasearch initiative as the basis for its searching profile, and SRU installations and implementations include those of the Library of Congress, Nature Publishing Group, Talis Information Systems and the Alvis project[5].

Summarising the key issues:

Target: search and information retrieval over the web, but primarily intended as a user-friendly replacement for the older Z39.50 protocol.

Advantages: Builds on the experience of the Z39.50 community from digital libraries.

Barriers: Information retrieval and digital libraries are not

[2] `http://www.loc.gov/sru/`
[3] `http://www.loc.gov/cql`
[4] e.g., YAZ, `http://indexdata.com/yaz/`
[5] `http://www.alvis.info`

strongly overlapping communities, where, for instance, evaluation of search results and the nature of content is quite different.

3.2 Metadata publishing

Two approaches provide metadata about resources.

3.2.1 OAI-PMH

The Open Archives Initiative Protocol for Metadata Harvesting[6] (OAI-PMH) is a flexible standard for allowing the publication and harvesting of *metadata about resources*. Examples of metadata are the well known Dublin Core (DC) used to described primarily publication data about a document. OAI-PMH servers are expected to provide DC metadata at a minimum, and may include other metadata such as digital rights. The resources described are intended to be books, physical media, web services and web content.

The OAI-PMH protocol provides commands for enquiries about available tags (i.e., tagged subsets which sub-groups can be retrieved by), and available metadata formats, and commands for retrieval of metadata selecting by date or by tags. The protocol operates via HTTP and returns results in XML with entries for the header and the metadata.

The primary users of this technology are libraries and digital libraries and publishing organisations within large institutions such as universities. Major search engines have embraced the protocol to interoperate with digital libraries.

Because the protocol allows arbitrary metadata formats, for instance, XSL transformations can be applied when publishing content, it can be used for more general XML publishing and harvesting tasks where meta-data content is a natural extension of the Dublin Core. For instance, semantic annotations, in-link information and categorisations could be included.

Summarising the key issues:

Target: Publication of meta-data about digital (and non-digital) resources. Intended as a means to support distributed systems.

Advantages: Bulk access to sets of metadata. Is used by search engines as a means of accessing some digital libraries.

Barriers: Distributed systems for IR have not been successful.

3.2.2 RSS

The Really Simple Syndication (RSS 2.0) standard provides a way to syndicate the content of a website in a push manner. RSS has some Dublin Core style fields and is widely supported. One extension for multimedia is the Media RSS proposal by Yahoo that may become a *de facto* standard for multimedia. It contains descriptors for bit-rate, sampling and so forth to properly describe audio and video content. Proper discovery of multimedia and its properties is a known problem for crawlers.

[6] `http://www.openarchives.org/OAI/openarchivesprotocol.html` `A9.com`.

The following subsection on OpenSearch also makes use of RSS. Note that in contrast to OAI-PMH which provides metadata in batches, RSS provides metadata about individual resources in a timely fashion, as they become available.

Summarising the key issues:

Target: Timely notification about digital resources. Targets internet news and blogging community.

Advantages: Solves the problem of crawling actively updated and produced content.

Barriers: Has seen rapid growth in use, and extensions are being proposed to extend its applicability to multimedia and search.

3.3 Search syndication: OpenSearch™

OpenSearch™ is a "set of simple formats for the sharing of search results." It consists of an XML resource definition for a search engine's capabilities, a results format, and a simple query language embedded in URLs.

OpenSearch is a standard proposed by Amazon to aggregate efforts around its search engine, A9.COM. It is an open format the uses a Creative Commons license. Like the SRU/CQL protocol it can operate via HTTP and returns results packaged in XML, in this case an extension of RSS 2.0, the syndication format.

The underlying idea is that every search service should be able to accept a standardized input format, and provide its results in another standardized format. This way, it is very easy to build services such as meta-search on top of basic search services. Search providers offer a description file that is an XML specification of their capabilities. This has terms such as `AdultContent` and `SyndicationRight` to describe functionality. A related idea is the meta-search standard STARTS [4], that did see use, perhaps because it did not have the support of a company such as `A9.com`.

The query syntax of OpenSearch, compared to CQL with its rich pedigree, is very simple: a list of keywords. Thus, this does not allow the flexibility of simple semantic-based search as in CQL. Moreover, the results format, an extension of RSS, does not provide a mechanism for providing more general aggregate information related to the query such as related categories or named entities.

The primary role of OpenSearch is to provide search syndication interoperability for search engines and aggregators. For this task, it has a well designed though simple schema. A remarkable number of search engines have enlisted with the capability. Support for it is built into some open source systems such as Nutch and DataparkSearch Engine. However, there is no clear result aggregation strategy, so it is unclear how well the standard can be used as a basis for distributed or federated search in any general sense.

Summarising the key issues:

Target: Smaller search engines, to allow syndication on

Advantages: Allows syndication of content.

Barriers: Lacks a more general strategy for results aggregation.

3.4 Site documents: Google Sitemaps

Sitemaps is a scheme Google is testing to inform and direct crawlers about available pages (for instance, in the hidden web or under difficult URLs) and about frequency of update. Google ties this with advantages to the website maintainer in terms of unique feedback about the site's search characteristics.

Sitemaps it not a *de facto* standard, but nevertheless it has been very well received by the community, especially due to the rise of dynamic web pages and frequently updated pages (e.g., blogs that update daily). Thus, this may become a *de facto* standard. Unlike RSS and OAI-PMH which advertise and aggregate metadata, Sitemaps merely provide, in a passive way, information to the crawler and thus represent a lower overhead for websites developers.

Summarising the key issues:

Target: Simplify crawling of sites, using Google site tools as a lure to get website maintainers involved.

Advantages: Makes crawling simpler.

Barriers: Seeing rapid adoption.

4. INFORMATION EXTRACTION

Recent developments in question answering systems, retrieval in XML, and semantic-based search share the common goal of offering more structured content to a user based on some underlying semantics recognised in the textual content, and possibly pre-tagged in the documents. A critical step for this in some applications is the use of information extraction (IE) or some other natural language processing to semantically tag documents. In some cases, such as the Wikipedia, basic tagging may already exist in the content, but in general some form of information extraction is required.

This general area, embedding more semantic information in content to support richer retrieval, is undergoing rapid development. Unlike indexing in current IR systems, there is no agreement on the right general architecture to employ this additional semantic information. Named entities and relations might be extracted and placed in a database, or some custom processing of hierarchical term spaces might be embedded in a retrieval engine. Full inference systems such as the open source Sesame system[7] for RDF are not currently practical for large document collections.

To support the information extraction step, in ALVIS we have developed an XML linguistic annotation format that supports this task [1], based on an emerging annotation format of the TC37SC4/TEI workgroup.

In most IR systems, linguistic processing is usually performed immediately prior to indexing time, but this restricts the processing to crude methods such as Porter stemming. In ALVIS we have also adopted an open, extensible architecture for document processing that allows components to be developed independently for different tasks in the document pipeline. One advantage of this approach is that XSL can be used for efficient extraction and conversion of content when document processing services with different needs communicate.

Other platforms for information extraction in the broader information access context include GATE [2] and UIMA [3], both mainly based on the Tipster format, and both implemented as Java systems which programs plug in to. Our architecture instead uses XML as the binding mechanism.

5. OTHER DEVELOPMENTS

In other areas, there is the potential for systems and standards to support open source information retrieval and search. here we have considered some of the major components of an information retrieval or search engine system, w.r.t. their suitability for standardisation.

5.1 Distributed search

Many paradigms exist for distributed search and information retrieval including Federated search engines with query routing (effectively acting like a meta-search system), and various forms of peer-to-peer. Peer-to-peer search works now in multimedia applications where the searchable content is short title strings. Federated search works in the digital library context where simple and effective results ranking strategies exist such as by date, location, etc.

For general information retrieval applications in a non-trivial distributed manner, there is no accepted methodology at present. If and when a practical methodology does emerge, standards should follow.

5.2 Crawling

Two crawl-related standards discussed previously were SiteMaps and OAI-PMH. The former eases the crawlers work at a site, and the later provides a means for crawlers to collect and redistribute resource meta-data in bulk. Crawling is a task that can be distributed more easily, on the Grub[8] system has on open source client-side for a distributed crawler, with an open protocol.

5.3 Personalisation

Personalisation is a task that would be well supported by an open standard with a transparent, secure, and trustworthy implementation. While one strategy is just to use Google for all one's desktop applications, stepping outside the monoculture requires personalisation be available to many different web applications and to roaming users. Moreover, a suite of common support tools for analysis need to be provided so that diverse systems can integrate personalisation.

5.4 Results ranking

Results ranking has two critical factors that make it particularly amenable to a standards based approach. The first factor is transparency. Offering access to ranking schemes

[7] `http://www.openrdf.org`

[8] `http:/www.grub.org`

and justifying ranking at runtime are commonly proposed as advantages of open source search.

However, transparency also leaves the potential for rank manipulation by, for instance, rank spamming methods. The second factor favoring open ranking is the potential to develop means of supporting the circumvention of this same rank spamming. This is an ideal community based task if appropriate trustworthy controls are enabled.

5.5 Static ranking

Static ranking is the ranking of documents independent of any query. PageRank^{TM} is a well known such ranking. It allows documents to be ordered within the collection to support, for instance, efficient results ranking. Static ranking can also be used on topic specific collections where different documents in the collection are more or less related to the topic of the collection.

5.6 Document storage

Why not standardise the storage of documents? As a very first step, documents could be stored in XHTML. The major problem is that HTML on the web is quite badly formed. Standard open source tools such as W3C's Tidy are still improving, and a common engineering approach is not to parse the HTML but instead to process it for word extraction, etc., basic tasks where parse trees are not required and robust tools exist to do partial parses on the fly.

5.7 Digital rights management

Digital rights management (DRM) is an essential feature to be integrated into search to broaden the pool of content, especially in the area of multi-media. This can already be seen in academic services such as Scirus[9], where some results are commercial content. Organisations such as the BBC and Deutsch Welle have large digital libraries which need protection if they are to be made available to the general public.

The Creative Commons provides a basis here for licensing, but it is not a digital rights management system. DRM standards need to be adopted by the large commercial organisations with the vested interests here.

6. CONCLUSION

A number of well developed standards, proposed standards and arguably *de facto* standards exist in the community, including CQL, SRU, OAI-PMH, Sitemaps, and OpenSearch.

CQL is best known in the digital library community, with traditions such as name spaces, fields, and data types such as dates. A lot of its functionality is shared by the specific query protocols adopted by IR systems such as Terrier and Lemur.

Sitemaps and OAI-PMH provide well thought out protocols for particular aspects of crawl and harvest. Both provide opportunity for use in the open source community beyond their original intension, with suitable extensions.

[9] `http://www.scirus.com`

Standards for tagging the results of information extraction (IE) are expected to see good use in open source IR because IE is a pipelined task that invariably requires a range of different tools, and the one tool can see common use on different IR systems. These two communities can interact well together through the use of standards.

6.1 Acknowledgments

The work is supported by the EU by the ALVIS STREP.

7. REFERENCES

[1] S. Aubin, J. Derivière, T. Hamon, A. Nazarenko, T. Poibeau, and D. Weissenbacher. A robust linguistic infrastructure for efficient web content ananalysis: the alvis project. In *Digital Semantic Content across Cultures*, Paris, 2006.

[2] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proc. of the 40th Anniv. Meeting of the Assoc. for Comp. Linguistics*, 2002.

[3] D. Ferrucci and A. Lally. UIMA: an architecture approach to unstructured information processing in a corporate research environment. *Natural Language Engineering*, 10:327–348, 2004.

[4] L. Gravano, K.-C. Chang, H. Garcia-Molina, and A. Paepcke. Starts: Stanford proposal for internet meta-searching (experience paper). In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 207–218. ACM Press, 1997.

Panel: IR Research and Open Source

Open Source Information Retrieval Workshop 2006

The panel brings together representatives from industry and academia to discuss their experiences with the development of open source tools as they pertain to information retrieval. The goal of this panel is to collect ideas for the coordination of research and the development of tools, data sets, and standards that can help spur future advances in the area. The experience of the panelists and their interaction with the audience should bring up new challenges and foster many new ideas and approaches. The panel is scheduled for the end of the Workshop to allow for the incorporation of the presented works in the discussions.