Efficiency Issues in Information Retrieval Workshop

European Conference on Information Retrieval - ECIR 2008
Glasgow, United Kingdom, 30 March 2008

Preface

Today's technological advancements have allowed for vast amounts of information to be widely generated, disseminated and stored. This exponentially increasing amount of information has rendered the retrieval of relevant information a necessary and cumbersome task. The field of Information Retrieval addresses this task by developing systems in an effective and efficient way. Specifically, IR effectiveness deals with retrieving the most relevant information to a user need, while IR efficiency deals with providing fast and ordered access to large amounts of information.

The efficiency of IR systems is of utmost importance, because it ensures that systems scale up to the vast amounts of information needing retrieval. This is an important topic of research for both academic and corporative environments. In academia, it is imperative for new ideas and techniques to be evaluated on as near-realistic environments as possible; this is reflected in the past Terabyte track and recent Million Query track organised by the Text REtrieval Evaluation Conferences (TREC).

In corporate environments, it is important that systems response time is kept low, and the amount of data processed high. These efficiency concerns need to be addressed in a principled way, so that they can be adapted to new platforms and environments, such as information retrieval from mobile devices, desktop search, distributed peer to peer, expert search, collaborative filtering, multimedia retrieval, and so on. Efficiency research over the past years has focused on efficient indexing, storage (compression) and retrieval of data (query processing strategies).

Some of the questions to be addressed in this workshop are: What are the efficiency concerns regarding IR applications (both new and traditional)? Do new applications create novel efficiency problems? Can existing efficiency related technology deal with these new applications? Has there been any advance in the last decade on state-of-the-art efficiency, or is it at a stand-still? Finally, bearing in mind past research on effciency, sketch future directions for the field.

Organisers

Roi Blanco – University of A Coruña, Spain
Fabrizio Silvestri – ISTI/CNR, Italy

Program Committee

- Alvaro Barreiro - University of A Coruña, Spain
- Christos Tryfonopoulos - Max-Planck-Institut für Informatik, Germany
- Aristides Gionis - Yahoo! Research Barcelona, Spain
- Ivana Podnar Zarko - FER, University of Zagreb, Croatia
- Justin Zobel - NICTA, Australia
- Antonio Gulli – Ask.com, Italy
- Hugh Williams - Microsoft Corporation, USA
- Massimo Coppola – ISTI/CNR, Italy
- Jie Lu - IBM Research, USA
- Ranieri Baraglia - ISTI/CNR , Italy
- Raffaele Perego – ISTI/CNR, Italy
- Fidel Cacheda - University of A Coruña, Spain
- Diego Puppin - Google Boston,USA

Program Editor

Roi Blanco – University of A Coruña, Spain

Web Site

Roi Blanco – University of A Coruña, Spain
http://irlab.dc.fi.udc.es/ecir

Sponsorship

The workshop is partially funded by: "Rede Galega de Procesamento da Linguaxe e Recuperacion de Informacion (Galician Network for Language Processing and Information Retrieval), funded by Xunta de Galicia".

Workshop Program

08:30 – 09:15	Registration
09:15 – 10:15	Keynote Speech Diego Puppin *The Index, the Cache and the Company's Cash: Daily Trade-offs for the Modern Information Retriever*
10:15 – 10:30	Q&A Session
10:30 – 11:00	Coffee Break
11:00 – 11:30	Judith Winter and Oswald Drobnik *A Distributed Indexing Strategy for Efficient XML Retrieval*
11:30 – 12:00	Vreixo Formoso, Fidel Cacheda and Víctor Carneiro *Algorithms for Efficient Collaborative Filtering*
12:00 – 12:30	Klaus Berberich, Srikanta Bedathur and Gerhard Weikum *Tunable Word-Level Index Compression for Versioned Corpora*
12:30 – 14:00	Lunch
14:00 – 14:30	Patrice Lacour, Craig Macdonald and Iadh Ounis *Efficiency Comparison of Document Matching Techniques*
14:30 – 15:00	David García-Pérez, Antonio Mosquera, Stefano Berretti and Alberto del Bimbo *Evaluation of a M-Tree in a Content-Based Image Retrieval System*
15:00 – 15:30	Benjamin Sznajder, Jonathan Mamou, Yosi Mass and Michal Shmueli-Scheuer *Metric Inverted - an Efficient Inverted Indexing Method for Metric Spaces*
15:30 – 16:00	Coffee Break
16:00 – 17:00	Discussion and Future Directions

Table of Contents

The index, the cache and the company's cash: daily trade-offs for the modern information retriever

Diego Puppin

Google, Boston, USA
diego.puppin@alum.mit.edu

Abstract. With more than one billion people accessing the Internet, with million queries asked daily, modern Web search engines are facing a problem of daunting scale. Search engines are expected to offer better and fresher results, faster, while Web is growing every day at accelerating speed. And all this must must be achieved while trying to limit the resources, the cash, put into crawling, indexing, querying and serving results.
Distributed search infrastructures, such as the document-partitioned and the pipelined term-partitioned architectures, are reaching a limit. A typical document-partitioned architecture has to scan the whole index before returning an answer, while term-partitioned architectures have major load-balancing problems.
We propose a new approach, based on representing documents as "bag of queries", which uses a document collection strategy to reduce the number of servers utilized by each query. By learning from users' behavior, our system can return very high quality results, with a strongly reduced computing load.
We also discuss the advantages and limits of the typical result cache as implemented in search engines, and we show how to exploit the peculiarity of our collection selection strategy to improve the result quality, while reducing the computing needs at the same time. Our technique is able to guarantee, for repeated queries, results comparable to those obtained with a centralized index, at a fraction of the cost.

A Distributed Indexing Strategy for Efficient XML Retrieval

Judith Winter[1] and Oswald Drobnik[1]

[1] J.W.Goethe University, Institute for Informatics, Frankfurt, Germany
{winter, drobnik}@tm.informatik.uni-frankfurt.de

Abstract. Using structural information from XML-documents can help to improve retrieval quality of these documents significantly, even in highly dynamic environments such as Peer to Peer (P2P) systems. However, due to the absence of a central index, the index information to be stored in a P2P network must be selected and distributed carefully to reduce communication costs and to guarantee scalability. We propose a novel indexing strategy for XML Information Retrieval in P2P systems. Indexing techniques to realize efficient retrieval include the use of highly discriminative keys (HDKs) which are created from rare combinations of content and structure information and used to support efficient retrieval, in particular for multi term queries. Depending on a peer's status, we index documents either globally into distributed indexes or locally in connection with distributing peer summaries. Finally, structural information from XML-documents is used to select entries for pruned posting and peer lists.

Keywords: Information Retrieval, Peer-to-Peer, XML Information Retrieval, XML-Retrieval, Indexing, Peer Selection, Highly Discriminative Keys, Distributed Search

1 Introduction and Related Work

Peer-to-Peer networks are a potentially powerful form of distributed computing, where peers – large sets of equal and autonomous nodes – are pooled together to share resources such as storage and computing power. Activities involved in search include issuing requests ("querying"), routing requests ("query routing"), matching results ("ranking"), and responding to requests ("retrieval") [8].

A precondition for efficient Information Retrieval (IR) is an adequate indexing strategy, which is the process of building an index over the document collection. Typically, an inverted index is used for storing the lexicon (index terms) and the posting lists (occurrences of the index terms in the collection) [2]. In P2P environments, two different methods of distributing an inverted index can be applied: document partitioning which is partitioning the document collection into several smaller sub-collections and indexing each of them, or term partitioning which is partitioning the inverted index by index terms [25].

Searching collections based on IR is a useful feature for the user. However, performing the search efficiently and maintaining scalability is one of the main obstacles in distributed environments [7]. Query processing consumes a considerable amount of resources even with centralized solutions; additional challenges arise in highly distributed P2P systems with bandwidth and latency constraints. The task of locating useful index information comes up, before relevant results can be ranked and retrieved. In order to reduce bandwidth consumption, P2P-IR algorithms aim at improving the efficiency of query evaluation by reducing the volume of index information fetched and processed, e.g. by using only selected information [12]. Selection techniques involve cutting off posting lists ("top-k pruning") or contacting only a small subset of participating peers that contain many documents relevant to the query ("resource selection") [8].

Approaches for Information Retrieval in P2P networks (P2P-IR) are a recent field of research. So far, acceptable response times are only achieved by solutions for simple pattern matching: documents are located in the P2P network by their name and only if they match the query exactly. An overview of P2P-IR approaches and criteria for their classification are presented in [17], whereas the issue of looking up data in P2P systems based on distributed hash tables (DHTs) is discussed in [3]. An efficient P2P search engine using IR techniques can be found in [15] where a key-based indexing strategy instead of single-term indexing is proposed, with keys being term sets that appear in a restricted number of documents, thus being highly discriminative keys. Two interesting approaches addressing the problem of efficiently selecting promising peers for answering a query are Minerva [4] and PlanetP [5]. Minerva is a distributed search engine on top of a DHT whose peer selection strategy is based on peer summaries. PlanetP is a content addressable publish/subscribe service for unstructured P2P communities that selects and contacts peers until k documents are found and until retrieving more documents would fail to contribute better results than these k documents.

The self-describing structure of XML can be a valuable source to achieve more precise and focused IR results when retrieving XML-documents. Methods include weighting diverse parts of documents differently; content and structure (CAS) queries enable users to specify structural constraints on what to retrieve; and retrieval units can consist of entire documents or only the most relevant parts of a document. A survey on indexing and searching XML-documents is conducted in [9]. Evaluation has shown successful application of XML Information Retrieval [10]. However, current approaches are all based on traditional client/server architectures.

Schema-based P2P networks or Peer Data Management Systems do apply XML in a P2P environment by providing techniques for the lookup of semi-structured data, e.g. by using XML P2P databases [23]. These systems consider exact or even partial matches and return either documents as a whole or XML fragments. However, they do not provide means to compute content relevance or structure similarity so far. A general idea about indexing, query routing, and query processing of XML data in a P2P network is presented in [6]. None of the analyzed approaches enables the use of IR techniques to compute relevance, though. The central challenge concerning the indexing process is identified as the handling of both value and path indexes. Deriving appropriate mappings of documents to peers is considered as the main problem of structured P2P systems. A fully self-organizing XML P2P database

system for sharing and querying XML data is presented in [19]. Information is requested with XQuery, so that a user can execute very expressive queries, but the results will have to match exactly.

To our knowledge, no P2P solutions for IR of XML-documents exist so far. Schema-based P2P-networks consider hints about the desired document structure but do not yet provide means to compute the relevance of documents. In this paper, we propose the first approach for XML Information Retrieval in a P2P system and concentrate on a novel indexing strategy for efficient retrieval.

2 Challenges for XML Indexing in P2P Systems

Approaches for ranking of XML-documents can incorporate structural information to improve retrieval quality. Additionally, retrieval units can be whole documents or document parts, therefore term statistics have to be collected for both. Even for centralized IR approaches, this introduces the challenge of storing the extra information efficiently in terms of storage space and access effort. Optimally, the information available for XML Information Retrieval would consist of all term statistics of all retrieval units in all XML documents for all possible query term combinations and would be accessible by all participating computers. However, this would result in a very exhaustive indexing process, especially if all indexing information is stored in a P2P network, with "churning" peers producing a highly dynamic data flow by joining and leaving the network. Hence, the information to be indexed and its distribution over the network have to be selected very carefully.

In P2P systems, the document collection to be queried is not static but undergoes constant change, e.g. by churning peers. In particular, there is no central index to store term weights and other evidence necessary for relevance computing, but the index is distributed over the network. Each newly indexed document leads to a bunch of messages sent between the peers to distribute the extracted information, and each churn of peers implies a redistribution of the existing information. Thus, communication overhead is a major cost factor [7]. The extra information stored and accessed for XML IR can additionally increase the amount of data transfer. Accordingly, the number and size of messages sent over the network should be minimized.

Due to the absence of a central index, peers cannot access required information directly but must locate it before use. Furthermore, the number of peers to be contacted for a specific query must be limited in order to guarantee scalability of the system – only carefully selected information can be used. Consequently, the index must be designed and distributed such that each peer can easily locate and access all content and structure information required for the peer's participation in answering a specific query. The peer has to store this information locally or must have knowledge how to get it from other peers. Information relates to the content of documents and document parts as well as their structure.

Around 85% of all queries are multi term queries that consist of more than one query term [14]. Especially in P2P systems, handling of multi term queries turns out to be a further challenge, since posting lists of all query terms must be matched with each other. This is of particular concern for large collections and popular terms, when

very large posting lists are transferred for matching leading to massive network traffic. In most existing P2P-IR approaches, much bandwidth is consumed by locating and sending large posting lists for the different query terms over the network, and expensive joins of these posting lists are performed at querying time. However, quick access to term combinations is desirable, especially to reduce network traffic and to facilitate query execution.

3 A Hybrid Indexing Strategy Based on HDKs

Our proposal for an indexing strategy aims at efficient querying by several techniques: pre-computing of posting lists for popular query term combinations; reducing posting list sizes to a fixed limit by indexing only rare terms (or term combinations) - highly discriminative keys; indexing documents either globally to distributed indexes or locally in connection with distributing peer summaries; and using structural information given by the user or by the documents themselves to select the entries that are stored in posting and peer lists.

3.1 Indexing Highly Discriminative Keys

Usually, documents are indexed per single terms. This might result in long posting lists for popular terms and in expensive joins of these posting lists for multi term queries. To avoid these problems, the use of highly discriminative keys was proposed in [15]. The original approach extracts all index terms of a document and distinguishes between rare and frequent keys. Only those keys whose global document frequency (*df*) with respect to the whole collection does not exceed a threshold are considered rare and specific enough to describe the document's content. Frequent keys, i.e. with a global *df* above that threshold, are regarded as non-discriminative. They are combined with each other if hints imply that they might be used together in a multi term query, e.g. based on a query log analysis [21] or if they occur in the same window of the same document. Only rare keys and rare key combinations are considered as highly discriminative keys and thus are indexed. As a result, the posting list of a HDK never exceeds the threshold. Multi term queries are supported by those HDKs that consist of several terms. Despite increased indexing costs by indexing several combinations per frequent key, the total traffic generated is notably smaller than when using a distributed single-term indexing strategies [16].

In extension of this approach, our HDK-based indexing strategy is applied to XML-documents, where structure can be taken into account at several steps of the retrieval process and consequently must be indexed. For each term in a document, structural information about the XML elements that contain this term is extracted together with the content. We denote by XTerm a tuple of content (i.e. term) and its structure, which is the path from the document root element to the term element in the XML-document tree expressed with XPath. The extracted XTerms are used to build HDKs by combining XTerms such that each combination is rare. A HDK can either consist of a single rare XTerm, in which case the XTerm's global frequency in the document collection does not exceed the frequency threshold. Or the HDK consists of

a set of frequent XTerms if each XTerm is frequent but the combination of XTerms is rare. We denote an index key as being either a HDK or an XTerm. The global frequency of an XTerm *t* is computed as a combination of the *df* of *t*'s content (as in the original HDK approach) and the *df* of *t*, i.e. different frequencies for different structures are taken into account. However, *df* of *t*'s content has more impact on the global frequency, as we focus on content-based search and use structure as vague hint considering that the user might not be able to specify the respective structure precisely.

While extracting and building the HDKs, their term statistics are collected for later ranking. These statistics are not limited to static document statistics but include information about document parts, too: for each document, evidence for several of its potential retrieval units is collected. The decision, which passages of a document are chosen as potentially good retrieval units, is based on several factors, e.g. the depth of a passage's path, the size of the passage, experience from past retrieval, parameters set by the user, or if the passage is very focused regarding a specific topic.

To support efficient retrieval, pruned posting lists are indexed for frequent XTerms in addition to full posting lists for HDKs. There are two reasons for this. First, HDKs composed of more than one XTerm have posting lists that contain only documents in which all XTerms appear: the posting list *pl(h)* for a HDK *h* is created as intersection of the posting lists of all XTerms t_i in *h*, i.e. $pl(h) = \cap\, pl(t_i \mid t_i \in h)$. Entries for documents that contain only one t_i are therefore not stored; this might be an undesired effect for highly relevant documents. Thus, the most relevant documents are stored for frequent XTerms, too, and can later be included into the ranking. Second, not all queries are composed of HDKs completely. When a given query *q* is executed, it has to be split into HDKs. The query terms would ideally form a single existing HDK – the posting list for the query would have been already computed and can be used directly after locating it. In case the query has to be split into several HDKs, it might happen that not all query terms can be covered and non-discriminative XTerms are left. In [24] it was proposed to either ask the user to specify such XTerms by adding further hints on content or structure, or to merge posting lists of HDKs containing those terms. However, this results either in additional effort for the user or in computation costs and network traffic for the posting list merging. Using an additional index for frequent XTerms can increase efficiency. The posting lists for frequent XTerms are pruned to a limit PL_{max} which is proportional to the frequency threshold used for HDKs so that these posting lists are limited to a fixed number of entries, too. The approach in [22], that extends the original HDK algorithm by applying query-driven indexing, suggests truncating posting lists –if necessary– by ranking the posting list entries according to the BM25 relevance computation scheme. We propose a more elaborated formula for pruning, such as the scoring function shown in formula (1) which is applied to choose the top *k* best posting list entries for XTerm *t*.

$$\begin{aligned} score_t(d_i) = \quad & \alpha * w(tf_t(d_i), icf_t) \\ & + \beta * w(tf_t(ru_{best}), icf_t) \\ & + \gamma * score_t(p_i) \end{aligned} \qquad \textbf{(1)}$$

The function is used to sort the posting list of *t* by $score_t(d_i)$ instead of simply using the term frequency as sorting key. This score is computed at indexing time and takes

into account weights for document d_i and its best retrieval unit ru_{best}., i.e. the retrieval unit with the highest weight for t. Alternatively, the average or even the sum of all indexed retrieval units of d_i could be considered. For the weighting function w, we adapted the BM25f formula [18] to XML IR. Parameters include the inverted collection frequency (*icf*) of t; the term frequency of t in d_i; and the term frequency of t in ru_{best}. The $score_t(p_j)$ depends on the quality of peer p_j on which d_i is stored. For instance, the peer score is high for peers with good collections regarding t and with good performance metrics such as response times. Values for α, β, and γ can be chosen such that $1 \geq \alpha \geq \beta \geq \gamma \geq 0$, i.e. the document weight will be favoured.

3.2 Hybrid Evidence Indexing

A peer can either index information locally or globally, depending on its status at indexing time. The more reliable and valuable a peer is, the more information it is allowed to send over the network (consuming more bandwidth and storage space) at indexing time, and the more information about the peer and its collection is available at querying time. A peer p is regarded as reliable, if it performed well in the past (e.g. provided highly relevant results for past queries) or if it provides a high quality collection (e.g. can achieve high scores regarding formula (1) for at least one t_i). Peer characteristics such as response times, available bandwidth, open IP address (vs. NAT-bound), latency, and CPU/Memory are also considered. To provide adequate peer statistics, we developed a P2P protocol that is based on Kademlia [11] and collects the desired information. Those peers will be preferred that have stayed online for at least x minutes, as the probability that a peer stays online increases with its uptime. For example, a network analysis of three file sharing systems found, that 65% of the peers joined the system online only once, that more than 20% of all connections lasted 1 minute or less, and that around 60% of the peers kept active no longer than 10 minutes each time they joined the system [20]. Of course, the peers' behaviour varies depending on the application. In our proposal, a relatively stable system is assumed where peers usually stay online over longer time periods. However, peers with short online times or which join the system only once and then disappear forever should not be allowed to perform an exhaustive indexing.

Our hybrid strategy distinguishes between peers with different status. Peers with full reliable status can store all extracted evidence into indexes distributed over the network, i.e. all term statistics are stored per document. Peers with low status (or if the user chooses "quick indexing") can store the extracted evidence only locally; a summary of the peer's overall collection will then be stored in the distributed indexes, i.e. all term statistics are stored per local collection. Hence, the indexing process distributes two lists for each key: a posting list with entries for documents (which where indexed per document), and a peer list with entries for peers (which hold documents that were indexed per collection). In the retrieval process, entries from both lists are taken into account, with a bias to posting list entries since these host the more significant information.

4 An Architecture Based on Distributed Indexes

Index information will be stored and accessed as shown in the architecture of each peer in figure 1. Black arrows denote local interaction between components of the same peer, whereas dotted arrows denote interaction between components of different peers via the P2P complex. The search engine is accessed by graphical user interface (GUI) which is part of the application complex. All interaction between components of different peers such as lookup and storage of indexed information are handled by the P2P complex.

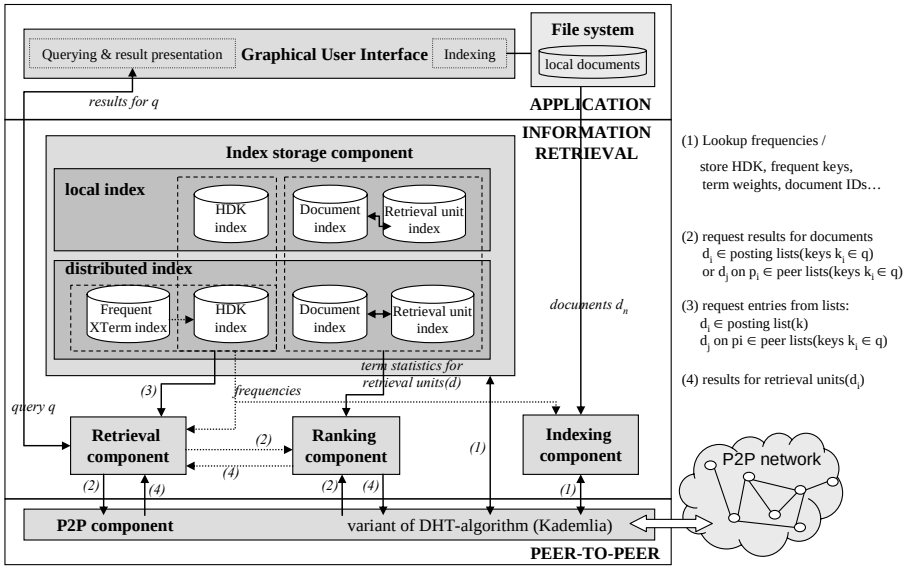


Fig. 1. A peer's architecture with several local and distributed indexes

In the following, we focus on the Information Retrieval complex which performs indexing, querying, and ranking of XML-documents. Both a term-partitioned and a document-partitioned strategy [25] are combined to store (and later retrieve) all indexing information: HDK index and frequent XTerm index act as term-partitioned inverted indexes by storing posting and peer lists of keys; document index and retrieval unit index hold the evidence used for relevance computing and are implemented both in a distributed clustered version and in a local version for subcollections.

The INDEXING COMPONENT parses XML-documents to extract all XTerms, creates HDKs from the extracted XTerms, and collects term statistics for potential retrieval units. For the HDK creation, global frequencies are requested via the P2P complex and received from the distributed HDK index and the frequent XTerm index. The following information is stored in local and distributed indexes:

- DISTRIBUTED HDK INDEX: For each HDK *h*, its global frequency, a posting list, and a peer list is stored. The posting list contains documents in which *h* occurs and is ordered by term frequency. These documents and their statistics are not stored locally but distributed among other peers. The peer list is ordered by peer score and contains peers that hold documents in which h occurs.
- LOCAL HDK INDEX: The same information as in the distributed version of this index is stored with exception of peer lists. All information relates to locally stored documents of the same peer.
- FREQUENT XTERM INDEX: For each frequent XTerm *t* with global *tf* exceeding threshold PL_{max}, a pruned posting list for the PL_{max} best documents and a pruned peer list for the best peers are stored. Links to all HDKs containing *t* are stored, too.
- DOCUMENT INDEX: For each document, statistics are maintained to compute the relevance of the document in the ranking process. Additionally, links to possible retrieval units of each document are stored. The statistics are represented by vectors to support ranking based on an extended vector space model.
  The local document index stores statistics of documents stored directly on the same peer (and indexed per collection). The distributed version of the index stores statistics of documents stored on other peers (and indexed per document).
- RETRIEVAL UNIT INDEX: Term statistics for the retrieval units of each indexed document are kept in vectors where each vector component represents the weight of an XTerm occurring in this retrieval unit.

In the retrieval process, the indexes are used as follows: Information from HDK index and frequent XTerm index is used by the retrieval component of a querying peer to decide which peers should participate in answering a given query *q*. Therefore, the query is sent to all peers with posting lists and peer lists for HDKs covering the query. The retrieval components of these peers will consult their lists to select promising peers and redirect the query to the selected peers. These peers hold statistics of potential relevant documents in their local document index or in their locally stored part of the distributed document index (and also can direct access the retrieval unit statistics). Once they receive the query, their ranking components will perform the ranking and send back retrieval results to the querying peer. In the GUI of the querying peer, a ranked list of links to relevant documents and retrieval units is presented to the user who can access them directly on the source peers.

5 Outlook

In this paper, a novel indexing strategy for efficient XML Information Retrieval in P2P systems has been proposed in order to achieve reduction of bandwidth consumption, support of multi term queries, and quick access to distributed information. The strategy is based on highly discriminative keys; posting lists for popular query term combinations are pre-computed at indexing time and their size is limited to a fixed threshold. Indexing is performed according to a hybrid strategy and depends on the indexing peer's status: only reliable peers can distribute exhaustive information; peers with an online time that is expected to be short can only store

evidence summaries. For the pruning of posting and peer lists, structural information is used to select the list entries. Finally, an architecture was designed that supports efficient retrieval by employing the proposed methods.

At present, we are implementing SPIRIX, a P2P search engine for Information Retrieval in XML-documents, on top of the centralized search engine Terrier [13]. A P2P protocol based on Kademlia has been developed that collects the peer statistics for the hybrid evidence indexing. For the ranking, we extended the vector space model with BM25F-oriented weighting to the use of XML IR by including weights for structural information. Experiments with our prototype to evaluate optimal parameters for the proposed algorithms will start soon.

References

[1] Amer-Yahia, S.; Lalmas, M.: ***XML Search: Languages, INEX and Scoring***. SIGMOD Rec. Vol. 35, No. 4, 2006.

[2] Baeza-Yates, R.; Castillo, C.; Junqueira, F.; Plachouras, V.; Silvestri, F.: ***Challenges on Distributed Web Retrieval.*** IEEE Int. Conf. on Data Engineering (ICDE07), Turkey, 2007.

[3] Balakrishnan, H; Kaashoek, F.; Karger, D.; Morris, R.; Stoica, I.: ***Looking Up Data in P2P Systems***. Communications of the ACM, Vol. 46, No. 2, 2003.

[4] Bender, M.; Michel, S.; Weikum, G.; Zimmer, C.: ***The MINERVA Project - Database Selection in the Context of P2P Search***. BTW Conference 2005, Karlsruhe, Germany, 2005.

[5] Cuenca-Acuna, F.; Peery, C.; Martin, R.; Nguyen, T.: PlanetP - Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In: Proc. of the 12th IEEE International Symposium on High Performance Distributed Computing, Washington, USA, 2003.

[6] Koloniari, G.; Pitoura, E.: ***Peer-to-Peer Management of XML Data: Issues and Research Challenges***. SIGMOD Rec. Vol. 34, No. 2, 2005.

[7] Li, J.; Loo, B.; Hellerstein, J.; Kaashoek, F.; Karger, D.; Morris, R.: ***On the Feasibility of Peer-to-Peer Web Indexing and Search.*** In: Proc. of the Second International Workshop on Peer-to-Peer Systems, 2003.

[8] Lu, J.; Callan, J.: ***Federated search of text-based digital libraries in hierarchical peer-to-peer networks.*** Information Retrieval Vol. 9, No. 4, Springer-Verlag, 2006.

[9] Luk, R.; Leong, H.V.; Dillon, T.; Chan, A.: ***A Survey in Indexing and Searching XML Documents***. JASIST, Vol. 53, No. 6, 2002.

[10] Malik, S.; Trotman, A.; Lalmas, M.; Fuhr, N.: ***Overview of INEX 2006.*** In: Proc. of the Fifth Workshop of the INitiative for the Evaluation of XML Retrieval, Germany, 2007.

[11] Maymounkov, P.; Mazieres, D.: ***Kademlia: A peer-to-peer information system based on the xor metric***. In Proceedings of IPTPS02, Cambridge, USA, 2002.

[12] Moffat, A.; Webber, W.; Zobel, J.; Baeza-Yates, R.: ***A pipelined architecture for distributed text query evaluation***. Springer Science + Business Media, LLC 2007.

[13] Ounis, I.; Amati, G.; Plachouras, V.; He, B.; Macdonald, C.; Lioma, C.: ***Terrier: A High Performance and Scalable Information Retrieval Platform***. In: ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006). Seattle, USA, 2006.

[14] Pass, G.; Chowdhury, Abdur; Torgeson, Cayley: ***A Picture of Search***. In: Proc. of First Int. Conference on Scalable Information Systems (Infoscale 2006), Hong Kong, 2006.

[15] Podnar, I.; Luu, T.; Rajman, M.; Klemm, F.; Aberer, K.: ***A Peer-to-Peer Architecture for Information Retrieval Across Digital Library Collections.*** In: Proc. of European conference on research and advanced technology for digital libraries (ECDL'06), Alicante, Spain, 2006.

[16] Podnar, I.; Rajman, M.; Luu, T.; Klemm, F.; Aberer, K.: ***Scalable Peer-to-Peer Web Retrieval with Highly Discriminative Keys***. In: Proc. of IEEE 23rd International Conference on Data Engineering (ICDE 2007), Istanbul, Turkey, 2007.

[17] Risson, J.; Moors, T.: ***Survey of research towards robust peer-to-peer networks – search methods***. In: Technical Report UNSW-EE-P2P-1-1, Uni. of NSW, Australia, 2004.

[18] Robertson, S.; Zaragoza, H.; Taylor, M.: ***Simple BM25 extension to multiple weighted fields***. In: Proc. of CIKM'04, ACM Press, New York, USA, 2004.

[19] Sartiani, C.; Manghi, P.; Ghelli, G.; Conforti, G.: ***XPeer: A Self-organizing XML P2P Database System***. Lecture Notes in Computer Science No. 3268, Springer-Verlag, 2004.

[20] Sen, S.; Wang, J.: ***Analyzing peer-to-peer traffic across large networks***. IEEE/ACM Transactions on Networking, Vol. 12, Issue 2, 2004.

[21] Skobeltsyn, G.; Aberer, K.: ***Distributed Cache Table: Efficient Query-Driven Processing of Multi-Term Queries in P2P Networks.*** ACM P2PIR'06, Virginia, USA, 2006.

[22] Skobeltsyn, G.; Luu, T.; Podnar, I.; Rajman, M.; Luu, T.; Aberer, K.: ***Web Text Retrieval with a P2P Quer-Driven Index.*** ACM SIGIR'07, Amsterdam, The Netherlands, 2007.

[23] Steinmetz, R.; Wehrle, K. (eds.): ***Peer-to-Peer Systems and Applications.*** Lecture Notes in Computer Science No. 3485, Springer-Verlag, 2005.

[24] Winter, J.; Drobnik, O.: ***Peer-to-Peer Cooperation for Content-Oriented XML-Retrieval.*** International Workshop on Peer-to-Peer Computing for Information Search (P2PSearch 2007), Jeju-Island, Korea, 2007.

[25] Zobel, J.; Moffat, A.: ***Inverted Files for Text Search Engines***. ACM Computing Surveys, Vol. 38, No.2, Article 6, 2006.

Algorithms for Efficient Collaborative Filtering

Vreixo Formoso[1], Fidel Cacheda[1], Víctor Carneiro[1]

[1] Department of Information and Communication Technologies, University of A Coruña
Facultad de Informática, Campus de Elviña s/n
15071 A Coruña, Spain
{vformoso, fidel, viccar}@udc.es

Abstract. In this work we present a series of collaborative filtering algorithms known for their simplicity and efficiency. The efficiency of this algorithm was compared with that of other more representative collaborative filtering algorithms. The results demonstrate that the response times are better than those of the rest (at least two orders of magnitude), in the training as well as when making predictions. Furthermore, when determining the quality of the predictions, the behavior of our algorithms is similar to that of the other algorithms, and even better when dealing with low-density training sets.

Keywords: Collaborative Filtering, Efficiency, Recommender Systems

1 Introduction

The amount of information found in the media, such as the Internet, is growing more and more each year, making it necessary to develop new forms of information retrieval (IR). Personalized retrieval systems are becoming more interesting, especially when not limited to just searching for information but that also are able to recommend the items that would be more appropriate for the user's needs or preferences.

Hence, the so-called recommender systems play an important role in the information society, thanks to their ability to predict the utility a particular item can have for a user, and thereby generate personal recommendations.

At present, recommender systems have proven to be useful in contexts such as e-commerce, and surely have a promising future in many others: Web search engines, program recommenders for digital TV, etc.

To achieve this personalization, recommender systems need to store certain information about the user preferences, known as the user profile. Depending on the way the information is obtained, we will distinguish between explicit and implicit systems. The former request the required information directly from the user, for example, by making the user evaluate an item using a numerical scale. In contrast, the implicit systems work in such a way that the user is unaware of its presence, obtaining information from the normal interaction between the user and the system. For

example, a search engine can obtain information about the user by following the searches he/she makes, the websites visited, etc.

There are two types of recommender systems, as a function of the algorithm used: content-based and collaborative filtering.

Content-based filtering selects what information should be recommended [5]. These systems present various limitations [23]:

- Item analysis should be performed automatically by a machine. This is problematic, especially when processing multimedia content, although the use of the multimedia contents mentioned has partially overcome this problem.
- It is unable to determine item quality. The quality of an item is a profoundly subjective characteristic dependent on the likes of each individual.
- It finds it difficult to describe "by chance" items that are interesting for the user (this is called serendipitous find).

In contrast, collaborative filtering systems [23] avoid these problems, given they are based not on item content but rather on the evaluation of other users. The system will inform the user of what items are well recommended by other users with similar likes or interests.

Because the evaluations are performed by individuals, an analysis of the content by the system is not necessary and the quality or subjective evaluation of the items will be considered. However, these algorithms present problems in their computational performance and efficiency.

In this work we present a series of collaborative filtering algorithms that are known for their efficiency, surpassing the response times of the more relevant collaborative filtering algorithms found in the literature. Moreover, the quality of the predictions made using the algorithms proposed is, at least, the same as that of other collaborative filtering algorithms.

In the section that follows, there is a brief introduction to collaborative filtering algorithms, including a brief description of the notation used throughout this work. After, there is a description of the collaborative filtering algorithms proposed. Section 4 presents details on how the algorithm evaluation was carried out, and finally, there is a summary of the conclusions and possible future lines of research.

2 Background

In collaborative filtering-based systems, user profile is a set of evaluations (explicit or implicit) carried out by the user on different items. This evaluation is usually represented as a numerical value on a particular scale, although it can also be unifying (indicating only those items of interest) or binary (indicating good and bad items) evaluations.

The system keeps a table containing the evaluations of the users, called an evaluation matrix. This table is processed to obtain the evaluation of new items, and thereby recommend new items to the users.

The way in which the data of this table are processed allows us to differentiate between two types of collaborative filtering algorithms: memory-based and model-based.

Memory-based algorithms carry out each prediction on the basis of the calculations performed using the entire table. Using similarity measurements the aim is to obtain users or items (called neighbors) that are similar to those for which we want to obtain a prediction, and calculate this prediction based on their neighbors. Most of these algorithms can be classified as user-based algorithms or item-based algorithms, depending on whether the process of obtaining neighbors was focused on obtaining similar users [18] [22] or items [20].

The model-based algorithms previously construct a model that is used to represent user behavior, and therefore, they make it possible to predict their evaluations. The parameters of the model are estimated offline using the data found in the table.

In the literature we find different approaches, most related with machine learning [13]: based on methods of linear algebra (SVD [2][19], Factor Analysis [4], PCA [6], MMMF [17], among others), clustering [25][11], graphical models, or techniques associated with artificial intelligence, such as Bayesian networks [3], latent class models [9][24] or neuronal networks [2].

Memory-based algorithms are simple compared to the model-based, but despite this, they give reasonably precise results. The model-based algorithms tend to be faster in prediction time than the memory-based, however, the construction of the model requires considerable time. Another advantage of model-based algorithms is their ability to find underlying characteristics in the data, this being very difficult to achieve in the memory-based system.

The main disadvantage of memory-based algorithms is their scalability, given each prediction made requires the processing of the entire table. With an elevated number of users or items, these algorithms are totally unadvisable for online systems that should recommend items in real time.

Furthermore, they are much more sensitive to various problems that are common in recommender systems than are the model-based:

- Sparsity: given most of the cells in the evaluation matrix are empty (without an evaluation) [20] [10].
- Cold-start: directly related with the previous point, there is the difficulty of making predictions for new users or items, and therefore, there are few evaluations [21].
- Spam: malicious users can apply certain techniques to influence the system [15].

However, despite these disadvantages, in practice the memory-based algorithms show the best results. The complexity of the models, dependent on multiple parameters, is difficult to foreseen and many times these are quite sensitive to changes in the data, long construction times for the model, or the problems of model updating when new date are available, make many algorithms unadvisable in a real system.

Precisely, to avoid this problem, some authors have focused on the development of algorithms that use models that are simple and fast to calculate [12], or that use model-based algorithm techniques as well as memory-based [16]

Finally, in an attempt to minimize the sparsity problem and the cold start, a third type of systems, called hybrids, was proposed. These combine the techniques of collaborative filtering with content-based methods [1][14].

2.1 Notation

The objective of collaborative filtering is the recommendation of a list of items to a user, or the prediction of the evaluation of a certain item. In both cases, the user object of the prediction or recommendation is known as active user.

In a typical scenario of collaborative filtering, there is a set of users $U=\{u_1, u_2, \ldots, u_m\}$ and another of items $I=\{i_1, i_2, \ldots, i_n\}$. Each user $u_i \in U$ has a profile, represented as a subset of items that have been evaluated, $I_u \subseteq I$, along with the corresponding evaluation for each item. Similarly, the subset of users that have evaluated a particular item, $U_i \subseteq U$, is defined. The active user is denoted as u_a.

A user evaluates each item giving it a score on a finite numerical scale. This set of possible evaluations is denoted as R.

Using the profiles of all the users we define the evaluation matrix, V, that represents their evaluations of the items. Each item of V, $v_{ui} \in R \cup \varnothing$, denotes the evaluation of the user $u \in U$ of item $i \in I$, the value $\varnothing$ indicating that the user had not yet evaluated the item.

Precisely, the objective of the collaborative filtering algorithm is to predict the value v in these cases. Let us denote $p_{ui} \in R \cup \varnothing$ as the prediction that the algorithm makes for the user's evaluation $u \in U$ for item $i \in I$. If the algorithm is unable to make this prediction, $p_{ui} = \varnothing$.

Finally, let us define the subset of evaluations carried out by a user, $v_{u\cdot} = \{v_{ui} \in V / i \in I_u\}$, and the group of evaluations for an item, $v_{\cdot i} = \{v_{ui} \in U / u \in U_i\}$. Let us denote $\overline{v_{u\cdot}}$ as the mean evaluation of a user, and as $\overline{v_{\cdot i}}$ the mean evaluation for an item. $\overline{v_{\cdot\cdot}}$ will be the overall mean evaluation.

3 Efficient Collaborative Filtering Algorithms

As recommender systems become more popular, they will have to deal with a higher number of users and items. In contexts such as Web IR, the potential number of users and information to treat is higher than in contexts such as e-commerce or recommendations of specific items (music, films...) in which traditionally, the use of collaborative filtering algorithms had been applied.

To make the leap to these new contexts, an algorithm should meet a series of conditions that presently they do not:

- Good behavior in settings of low density. The sparsity problem, present in domains of limited reach (such as film recommenders, for example), become more serious as the information is diversified. In these contexts, the problem is not only motivated by a high number of available items, but also because they belong to very different domains.
- Computational efficiency. The algorithm should be scalable, to be able to handle the volume of information and of users of the system present in

contexts such as IR. Memory-based algorithms, for example, require an extensive calculation that makes them difficult to scalable to these new needs.

- Constant updating. Many of the model-based algorithms are based on the more or less static nature of the data. The construction of the model, expensive computationally, is carried out offline and every certain amount of time. In certain contexts, this supposition is more or less correct. However, contexts such as Web IR are intrinsically dynamic, with continuous entries, withdrawals and modifications of the information.

Next, a series of collaborative filtering memory-based algorithms designed for Web IR is presented, with special emphasis on their computational efficiency.

3.1 Item Mean Algorithm

This algorithm can be considered the baseline, given it consists in taking the mean of an item as prediction of its evaluation. Thus, $p_{ui} = \overline{v_{\cdot i}}$.

It is based on the fact that if an item is a good recommendation for many people, and therefore it has an elevated mean, one can assume that it has many possibilities of being a good recommendation for this user. The idea is to recommend items that are generally considered good.

However, this algorithm does not take into account user- or item-dependent variations, which could negatively influence the quality of the recommendations.

3.2 Simple Mean Based Algorithm

This algorithm is also based on the mean evaluation of an item, but corrected according to the mean of the user.

The idea is to try to take into account, in a very simple manner, the way a user emits his/her evaluations. In fact, the only aspect that will be considered is going to be the tendency of the user to evaluate positively or negatively, following the formula:

$$p_{ui} = \overline{v_{\cdot i}} + \frac{\sum_{j \in I_u} (v_{uj} - \overline{v_{\cdot j}})}{|I_u|}$$

The idea is the capture the variation between the mean of an item and the evaluation of the user, correcting the mean of the item to predict with this variation.

3.3 Tendencies Based Algorithm

This algorithm expands the previous idea, keeping in mind the mean of the users and the items, along with the variations that affect each user or item in particular.

Starting with the fact that the users evaluate the items differently [18], we propose to capture the tendency of the user. Thus, it should be determined if a user has the tendency to evaluate the items positively, or on the contrary, to evaluate them negatively.

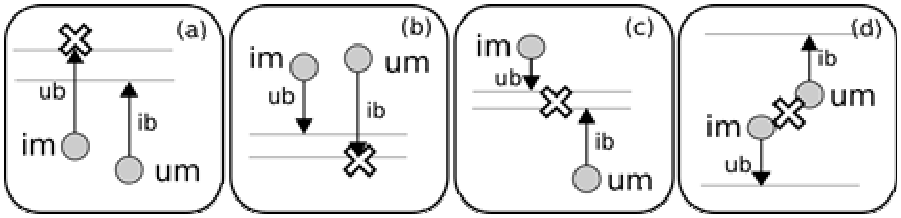


Fig. 1. Relationship between the means (circles) and the tendencies (arrows). *im* and *um* represent the means of the items and user, respectively. *ib* and *ub* represent the tendencies of the item and the user, respectively.

It is important to not confuse this tendency with whether the mean of the user is high or not. For example, a user that only evaluates items that he/she has liked will have a high mean, but it is possible that the evaluations are lower than the mean of each item. Thus, the user tends to evaluate the items negatively, even despite their high mean.

Therefore, we define the tendency of a user (ub_u) as the mean difference of his/her evaluations with respect to the mean of the item.

$$ub_u = \frac{\sum_{i \in I_u} (v_{ui} - \overline{v_{\cdot i}})}{|I_u|}$$

We are also interested in capturing the tendency of an item (ib_i), that is, if the users consider it to be an especially good or especially bad item. In this case, the aim is not to determine if the item is well evaluated, but rather to see if it stands out among the items evaluated by a user. As in the previous case, we are interested in the relative evaluations (the evaluation with respect to the mean of the user), and not the absolute mean of the item.

$$ib_i = \frac{\sum_{u \in U_i} (v_{ui} - \overline{v_{u\cdot}})}{|U_i|}$$

The algorithm proposed takes into consideration both the mean of the user and item, as well as of their respective tendencies when computing a prediction. There are various cases that depend on these values, as depicted in Fig. 1.

In the first case, Fig. 1 (a), both the user as well as the item have a positive tendency, that is, the user tends to evaluate the items above their mean, and the item tends to be evaluated above the mean of the user. Hence, the prediction will take a value that will be above the mean of both:

$$p_{ui} = \max(\overline{v_{u\cdot}} + ib_i, \overline{v_{\cdot i}} + ub_u)$$

where the use of the maximum tends to give a better evaluation to this type of items, whose tendency indicates that they are good.

The second case, Fig. 1 (b), is the opposite situation: both the user and the item have a negative tendency, that is, the user tends to evaluate the items below their mean, and the item tends to be evaluated below the mean of the user. In this case the prediction is computed as:

$$p_{ui} = \min(\overline{v_{u\cdot}} + ib_i, \overline{v_{\cdot i}} + ub_u)$$

where the use of the minimum has to make sure that the item, whose tendency indicates that it is a bad recommendation, is not recommended simply because the user had a very high mean.

The third case, Fig. 1 (c), is when we find a "negative" user (its tendency is to evaluate the items with values below its mean), and a good item (its tendency is to be evaluated above the user mean), or vice versa.

If the means of both corroborate their tendencies (that is, user with low mean and item with high mean), the prediction will be in the middle between both, closer to one or the other depending on the value of the distinct tendencies. In this case, the prediction is computed as:

$$p_{ui} = \min[\max(\overline{v_{u\cdot}}, \overline{v_{\cdot i}} + ub_u)\alpha + (\overline{v_{u\cdot}} + ib_i)(1-\alpha)), \overline{v_{\cdot i}}]$$

where α is a parameter that allows to grant a greater confidence in the mean of the user or of the item. Empirically, we have proven that values slightly about 0.5 (0.6, 0.65) gave the best results.

Finally, it is possible for the means to not corroborate the tendency, Fig. 1 (d). A user with negative tendency evaluates an item of low mean (expecting the prediction to be bad), but at the same time the mean of the user is high and the tendency of the item is positive (that on the contrary would indicate at good evaluation). In this case, the prediction is calculated as:

$$p_{ui} = \overline{v_{\cdot i}}\alpha + \overline{v_{u\cdot}}(1-\alpha)$$

This case seems to contradict the suppositions of our algorithm, hence, it will be analyzed more thoroughly in the next section.

4 Evaluation

The evaluation of the algorithms proposed aims to study their efficiency, in comparison with other collaborative filtering algorithms (Section 4.2). However, we also include in Section 4.3 a brief comparative of the quality of the predictions made using the different algorithms.

4.1 Methodology

The different algorithms have been evaluated using the DataSet MovieLens [7]. This dataset contains real data corresponding to the rating of films by users, captured on the website of the film recommender MovieLens (http://movielens.umn.edu), during a period of 7 months (from 19-09-1997 to 22-04-1998). From these data users showing less than 20 evaluations were eliminated, obtaining a total of 100,000 evaluations, from 943 users for 1,682 movies. Therefore, the proportion between users and items is 1.78 items per user, and the density is 6%. The evaluations are discrete and take values between 1 (low evaluation) and 5 (high evaluation).

The algorithms proposed are compared with a selection of collaborative filtering algorithms:

- Memory-based:
    - User-based [8] [18].
    - Item-based, using the Adjusted Cosine Similarity as similarity measure, when the best results are obtained [20].
    - Similarity fusion [27].
- Model-based:
    - Regression based [26].
    - Slope one [12]
    - Latent Semantic Indexing (LSI) [19]
    - Cluster based smoothing [28]
- Mixed:
    - Personality diagnosis [16].

The trials were performed by dividing the dataset into two groups: a training subset and an evaluation subset. The training subset is constructed by selecting a percentage of the evaluations available in the dataset. The remaining evaluations will be part of the evaluation set. The training subset is used by the algorithm to predict the value of the data from the evaluation subset.

To evaluate the quality of the predictions, the evaluation obtained from the algorithm is compared with the original evaluation present in the evaluation subset.

In the experiments performed we are working with training sets of various sizes to evaluate performance in situations of very low (10%), intermediate (50%) and high (90%) densities.

For each algorithm evaluated, the training and prediction times, as well as the quality of their predictions are measured. For a greater confidence in the results obtained, each trial was repeated 5 times, then taking the mean value.

All the experiments were performed on an AMD Athlon(tm) 64 Processor 3200+, 1 GB RAM and 150 GB HDD. All the algorithms have been implemented in Java and executed on the machine with no type of charge. The dataset is stored in database MySQL 5.0.38 in the same computer.

With respect to case (d) from Section 3.3, we have verified that this case occurs in less than 5% of predictions, which confirms the good behavior of the tendencies system. Secondly, we have found that as the density of the dataset increases, this percentage is reduced even further, reaching 2% when using the training set at 90% of the evaluations available (equivalent to density of 6%). In summary, this case is produced mainly when the tendencies are based on very few evaluations, and even in this case they are minimums. This finding reinforces the idea that the tendencies versus the mean are a good reflection of reality and that they can be a good prediction mechanism.

4.2 Efficiency

To evaluate the efficiency of the algorithms proposed, a theoretical study and an empirical analysis were performed. In the theoretical study, the computational complexity of the different algorithms evaluated for training (or model construction) and for prediction was determined.

Table 1. Computational complexity of the collaborative filtering algorithms. The number of users is denoted as *m* and the number of items is denoted as *n*.

Algorithm	*Training complexity*	*Prediction complexity*
User Based	-	O(mn)
Item-Based	$O(mn^2)$	O(n)
Similarity Fusion	$O(n^2m + m^2n)$	O(mn)
Personality Diagnosis	$O(m^2n)$	O(m)
Regression Based	$O(mn^2)$	O(n)
Slope One	$O(mn^2)$	O(n)
Latent Semantic Indexing	$O((m+n)^3)$	**O(1)**
Cluster Based Smoothing	$O(mn\alpha + m^2n)$	O(mn)
Item Mean	**O(mn)**	**O(1)**
Simple Mean Based	**O(mn)**	**O(1)**
Tendencies Based	**O(mn)**	**O(1)**

Table 1 shows the complexities for the different algorithms, indicating in bold the algorithms with the best computational efficiency. As observed, the algorithms proposed showed a lower computational complexity for training and prediction.

Table 2 shows the results of the empirical analysis. In this case, the running times of the different algorithms were measured during training and when making predictions, considering training groups 10%, 50% and 90% of the size of the dataset (respectively, 10,000, 50,000 and 90,000 evaluations), and the rest as evaluation set.

With respect to training time, the user-based algorithm showed the best results given it lacks a training period, followed by the algorithms proposed. Overall, any of the algorithms proposed present training times that are at least two orders of magnitude lower that the rest of the collaborative filtering algorithms.

For prediction time, the times in Table 2 refer to the time needed to calculate the predictions: 90,000, 50,000 and 10,000, for the 10%, 50% and 90% training sets, respectively.

Table 2. Training time and prediction time for the collaborative filtering algorithms, as a function of the size of the training set: 10%, 50% and 90 % (respectively, 90%, 50% and 10% for the prediction set). Time units are in milliseconds.

Algorithm	*Training time*			*Prediction time*		
	10%	*50%*	*90%*	*10%*	*50%*	*90%*
User Based	0	0	0	6,250	15,597	8,915
Item Based	415	1,060	1,986	221	1,864	909
Similarity Fusion	987	3,840	5,474	227,736	756,834	264,951
Personality Diagnosis	257	994	2,213	1,369	3,845	1,400
Regression Based	3,302	4,575	7,780	205	570	265
Slope One	1,246	2,175	2,541	319	501	116
Latent Semantic Indexing	117,758	115,218	102,855	162	158	20
Cluster Based Smoothing	60,247	71,529	44,635	70,515	251,595	118,552
Item Mean	2	3	3	24	12	2
Simple Mean Based	7	10	5	25	11	4
Tendencies Based	11	15	9	24	16	4

In this case, the algorithms proposed are clearly the most efficient. The user-based algorithm, that presented good behavior in the training, shows a prediction time that is three orders of magnitude higher. The Latent Semantic Indexing algorithm, showing good behavior, is one order of magnitude slower and its training time is the worst one from the algorithms analyzed.

Overall, one would expect that when you increase the number of predictions the prediction time would also increase in proportion. Thus, with a training group at 10% (equivalent to 90,000 predictions) the time is lower than that with a training group at 50% (50,000 predictions), which seems contradictory. This is due to the fact that in situations of very low density the information available is minimal and it is not possible to carry out all the predictions of the predictions group. The predictions that cannot be made use up much less time, and the overall time of prediction is reduced.

Lastly, and without going into a detailed analysis, the results show that the theoretical study of the computational complexities corresponds with the times obtained for the training and the predictions.

4.3 Prediction quality

In the previous section the efficiency of the algorithms presented in this work was demonstrated. However, it is also necessary to determine the quality of the algorithms proposed.

In this sense, Table 3 presents the Mean Absolute Error (MAE) of the different algorithms evaluated as a function of size of the training set. As expected, as the training set increased, the results of the algorithms improved, producing fewer errors.

Focusing on the algorithms proposed (Simple Mean Based and Tendencies Based), we can observe that in situations of very low density (training set at 10%) the best results are obtained, and in the other cases (50% and 90%) their results are similar to those of the best algorithms. So, the algorithms proposed show stable behavior, making good predictions with a high number of evaluations (like the rest of algorithms), but also when the number of evaluations is reduced (e.g. cold start and sparsity problems).

Table 3. Mean Absolute Error of the collaborative filtering algorithms, as a function of the size of the training group: 10%, 50% and 90 % (respectively, 90%, 50% and 10% of the training group).

Algorithm	*MAE*		
	10%	*50%*	*90%*
User Based	0.99	0.71	0.68
Item-Based	0.92	0.75	0.71
Similarity Fusion	0.84	0.73	0.71
Personality Diagnosis	0.82	0.78	0.78
Regression Based	1.03	0.76	0.74
Slope One	0.90	0.72	0.70
Latent Semantic Indexing	0.85	0.77	0.73
Cluster Based Smoothing	0.97	0.87	0.80
Item Mean	0.82	0.79	0.79
Simple Mean Based	0.79	0.72	0.72
Tendencies Based	0.79	0.72	0.71

5 Conclusions

In this work we have presented a series of algorithms that, based on the simplicity, obtain response times that are better than those of the algorithms evaluated. Moreover, when evaluating the quality of the predictions, our algorithms present a behavior that is equivalent to that of the best algorithms, and even improving the results with very low density training groups.

Keeping this in mind, it would be interesting to use these algorithms in real settings, especially in Web IR systems.

Along this line, future works are aimed towards the adaptation of these algorithms to a search engine. The objective is to improve the quality of the results and make it easier for the user to find relevant information, probably already found previously by other similar users.

Acknowledgments
This work was partially supported by the Spanish government under project TSI2005-07730.

References

1. Basilico, J. and Hofmann, T.: Unifying collaborative and content-based filtering. In Twenty-First International Conference on Machine Learning. vol. 69. ACM, New York, NY, 9. 2004.
2. D. Billsus and M. J. Pazzani.: Learning collaborative information filters. In Fifteenth International Conference on Machine Learning, pp 46-54, Madison, WI, Morgan Kaufman, 1998.
3. Breese, J. S., Heckerman D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In Fourteenth Annual Conference on Uncertainty in Artificial Intelligence, pq 43-52, July 1998.
4. Canny, J.: Collaborative filtering with privacy via factor analysis. In 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, New York, NY, pp. 238-245, 2002.
5. Foltz, P. W., Dumais, S. T.: Personalized information delivery: an analysis of information filtering methods. Communications of the ACM, Volume 35, Issue 12. Special issue on information filtering. pp: 51 – 60. (December 1992)
6. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A Constant Time Collaborative Filtering Algorithm. Information Retrieval 4, 2, pp. 133-151. 2001.
7. Herlocker, J., Konstan, J., Borchers, A., Riedl, J.: An Algorithmic Framework for Performing Collaborative Filtering. In 22th Annual International ACM SIGIR Conference on Research and Development in information Retrieval. pp. 230-237, 1999.
8. Herlocker, J. L., Konstan, J. A., Riedl, J.: An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. Information Retrieval, 5, pp. 287–310. 2002.
9. Hofmann, T., Puzicha, J.: Latent Class Models for Collaborative Filtering. In Sixteenth International Joint Conference on Artificial intelligence. T. Dean, Ed. Morgan Kaufmann Publishers, San Francisco, CA, 688-693. 1999.

10. Huang, Z., Chen, H., Zeng, D.: Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. ACM Transactions on Information Systems 22, 1, pp. 116-142. 2004.
11. Kohrs, A., Merialdo, B.: Clustering for collaborative filtering applications. In Computational Intelligence for Modelling, Control & Automation. IOS Press, 1999.
12. Lemire, D., Maclachlan, A.: Slope one predictors for online rating-based collaborative filtering. In SIAM Data Mining, 2005.
13. B. Marlin. Collaborative filtering: A machine learning perspective. Master's thesis, University of Toronto, 2004.
14. Melville, P. Mooney, R.J., Nagarajan, R.: Content-boosted collaborative filtering. In ACM SIGIR Workshop on Recommender Systems, Sep 2001.
15. Mobasher, B., Burke, R., Bhaumik, R., Williams, C.: Effective Attack Models for Shilling Item-Based Collaborative Filtering System. In WebKDD Workshop (KDD'2005). Springer, Chicago, Illinois, USA (2005)
16. Pennock D.M., Horvitz, E. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In IJCAI Workshop on Machine Learning for Information Filtering, International Joint Conference on Artificial Intelligence. 1999.
17. Rennie, J. D., Srebro, N.: Fast maximum margin matrix factorization for collaborative prediction. In 22nd international Conference on Machine Learning ICML '05, vol. 119. ACM, New York, NY, 713-719. 2005.
18. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: GroupLens: An Open Architecture for Collaborative Filtering of Netnews, Internal Research Report, MIT Center for Coordination Science, March 1994.
19. Sarwar, B., Karypis, G., Konstan, J., and Reidl, J.: Application of dimensionality reduction in recommender systems--a case study. In ACM WebKDD Workshop, 2000.
20. Sarwar, B., Karypis, G., Konstan, J., and Reidl, J. 2001. Item-based collaborative filtering recommendation algorithms. In 10th international Conference on World Wide Web. WWW '01. ACM, New York, NY, 285-295. 2001.
21. Schein, A. I., Popescul, A., Ungar, L. H.: Methods and Metrics for Cold-Start Recommendations. In 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, New York. 2002
22. Shardanand, U.: Social Information Filtering for Music Recommendation, S.M. Thesis, Program in Media Arts and Sciences, Massachusetts Institute of Technology, 1994.
23. Shardanand, U., Maes, P.: Social Information Filtering: Algorithms for Automating "Word of Mouth". In ACM CHI'95 Conference on Human Factors in Computing Systems. ACM, New York. 210–217. 1995.
24. Si, L., Jin, R.: A Flexible Mixture Model for Collaborative Filtering, In 20thInternational Conference on Machine Learning (ICML 2003). 2003.
25. Ungar, L.H., Foster, D. P.: Clustering Methods For Collaborative Filtering. In Workshop on Recommendation Systems. AAAI Press, Menlo Park California. 1998.
26. Vucetic, S., Obradovic. Z.: A regression-based approach for scaling-up personalized recommender systems in e-commerce. In ACM WebKDD 2000 Web Mining for E-Commerce Workshop, 2000.
27. Wang, J., de Vries, A. P., Reinders, M. J.: Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In 29th Annual international ACM SIGIR Conference on Research and Development in information Retrieval SIGIR '06. ACM, New York, NY, 501-508. 2006.
28. Xue, G., Lin, C., Yang, Q., Xi, W., Zeng, H., Yu, Y., Chen, Z.: Scalable collaborative filtering using cluster-based smoothing. In 28th Annual international ACM SIGIR Conference on Research and Development in information Retrieval SIGIR '05. ACM, New York, NY, 114-121. 2005.

Tunable Word-Level Index Compression for Versioned Corpora

Klaus Berberich, Srikanta Bedathur, and Gerhard Weikum

Max-Planck Institute for Informatics, Saarbrücken, Germany
{kberberi,bedathur,weikum}@mpi-inf.mpg.de

Abstract. This paper presents a tunable index compression scheme for supporting time-travel phrase queries over large versioned corpora such as web archives. Support for phrase queries makes maintenance of word positions necessary, thus increasing the index size significantly. We propose to *fuse* the word positions in many neighboring versions of a document, and thus exploit the typically high level of redundancy and compressibility to shrink the index size. The resulting compression scheme called FUSION, can be tuned to trade off compression for query-processing overheads. Our experiments on the revision history of Wikipedia demonstrate the effectiveness of our method.

1 Introduction

Nowadays more and more digital content is archived, thus preserving it for the future. Documents in these collections are either explicitly versioned – as in the case of corporate documents or wiki pages, or timestamped, as in web archives, where timestamps reflect the time of discovery. Access to these collections is often highly restricted, and a search functionality, when available, ignores the temporal dimension of the collection.

Search over versioned corpora such as web archives and wikis has recently attracted some interest from the research community, as can be seen from several recent related publications [1–4]. Exploiting the typically high level of content redundancy in these collections is a common theme in all of them. So-called *historical queries* or *time-travel queries* are covered by Anick and Flynn [1] and in our previous work [2]. Given a user-specified time point or time interval of interest, the query is evaluated taking into account only document versions that existed back then. In our previous work [2] we described techniques that provide efficient support for *ranked keyword queries*, but did not consider *phrase queries*.

Phrase queries are valuable whenever the user is looking for person names, movie titles, products, and other entities that can be identified unambiguously by a sequence of words. In order to support phrase queries, word positions must be kept in the index, which inflates the index. This blow-up is typically countered by compressing the positions of each word within a document by taking differences (d-gaps) and using a space-efficient coding scheme (e.g., Elias or Golomb codes).

Changes between document versions tend to be small, leaving large portions of the document unchanged. As a consequence, the set of positions in consecutive document versions of a word often either (i) overlap significantly (e.g., if content is appended) or (ii) are *numerically close* (e.g., if only a few words are inserted/deleted).

Problem Statement In this work, we address the problem of supporting *time-travel phrase queries*. We assume that an inverted index is employed, i.e., for each word w its occurrences in the corpus are organized in a posting list. Given a sequence of timestamped document versions d^{t_i} containing the word w, we seek to find a compact lossless representation of w's positions.

Contributions We introduce an effective solution to the above problem. The method builds postings covering multiple consecutive document versions, thus yielding a significant reduction of the overall data volume. Since these postings can be larger than postings for single document versions, possibly more data must be read at query-processing time. The proposed method is tunable and allows trading-off the resulting query-processing overheads and overall compression. Further, it allows plugging in arbitrary schemes to compress postings. Its effectiveness is demonstrated in an experimental evaluation using the revision history of the English Wikipedia as a real-world large-scale versioned corpus.

Organization An overview of relevant related work is given in Section 2. In Section 3 we illustrate the ideas behind our approach, before describing the underlying optimization algorithm in Section 4. Following that, we describe the conducted experimental evaluation in Section 5, and finally conclude the present work in Section 6.

2 Related Work

Recently, search over document collections that (i) are versioned or (ii) exhibit a high level of redundancy has attracted some interest from the research community. The earliest proposal by Anick and Flynn [1] describes a help-desk system that supports historical queries by essentially indexing delta change records. Broder et al [5] propose a method that avoids indexing shared content. Their method, though, is applicable only to collections with "well-behaved" content overlap such as E-mail conversations. Hersovici et al. [3] yield a more compact index by exploiting redundancy in a sequence of multiple document versions. They determine and index so-called virtual documents containing content that is common in a subsequence of the document versions. Zhang and Suel [4] exploit the redundancy in the collection by indexing text fragments instead of complete documents. Additional bookkeeping is then required to keep track of the fragments contained in a document. In our own previous work [2], we exploit redundancy and thus compact the index by coalescing postings belonging to consecutive document versions that bear almost-identical payloads. Further, the approach can systematically trade-off index size for query-processing efficiency by decomposing posting lists into sublists covering small portions of the time axis. Only Anick and Flynn [1] and our previous work [2] deal with so-called historical or time-travel queries, where the idea is to search on the collection "as of" a user-specified time. None of the approaches mentioned, however, provides a satisfying solution for phrase queries.

Compression of indexes is a well-studied topic in information retrieval, and has produced a wealth of publications. Here, the core problem is to compactly represent an

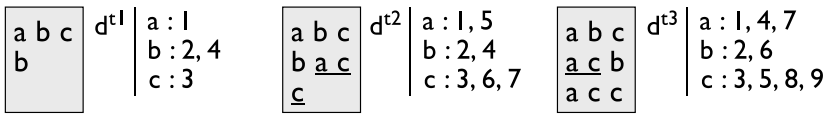


Fig. 1. Three timestamped document d^{t_1}, d^{t_2}, and d^{t_3} containing words a, b, and c.

increasing integer sequence of document identifiers or word positions. Typically, the sequence is first converted into differences, and then encoded using a space-efficient coding scheme. Elias' γ-codes and δ-codes are two popular non-parameterized, Golomb codes and Golomb-Rice codes two popular parameterized coding schemes. For detailed descriptions of these coding schemes, we point to Witten et al. [6]. Their applicability to the inverted index is discussed by Moffat and Zobel [7], a thorough comparison of different implementation alternatives can be found in Scholer et al. [8].

3 Compressing Word Positions

Word positions are typically organized in an *inverted index* – the standard index structure for large-scale text search. Words that occur in the collection are kept in a *lexicon* organized, for instance, as a B+-Tree. For each word the lexicon points to its corresponding posting list. These *posting lists* are often organized as flat lists, but can also be organized in a more sophisticated manner, for instance, if efficient support for time-travel queries [2] is needed. For the scope of this work, we assume that an inverted index is employed, i.e., for a particular word, the information about its occurrences is kept in a posting list, we do, however, not make further assumptions about the internal organization of these posting lists.

3.1 Baseline

We now describe a baseline solution that keeps word positions per document version d^{t_b} per contained word in a separate posting having the structure

$$\langle \quad [\, t_b,\, t_e\,) \;\mid\; did \;\mid\; positions \quad \rangle\,.$$

The first component $[\, t_b,\, t_e\,)$ is the *validity time-interval* of the document version conveying when the document version existed. For document version d^{t_i} the validity time-interval is defined as $[t_i,\, t_{i+1})$, if a newer version exists, and $[t_i,\, now)$ otherwise. The special value now always points to the current time. Keeping the validity time-interval, instead of only the document version's timestamp, is necessary to support time-travel queries efficiently. The document identifier did is kept in the second component. The third component maintains the list of word positions.

Depending on implementation choices, each of the three components is represented compactly using one of the space-efficient compression schemes mentioned in Section 2.

In our concrete implementation, validity time-interval boundaries and document identifiers are stored as uncompressed 64 bit integers, positions are compressed by taking their differences and encoding them using Elias γ- or δ-codes. For the problem instance given in Figure 1, we obtain the following posting for document version d^{t_1} and word b

$$\langle \ [t_1, t_2) \mid d \mid 2, 4 \ \rangle .$$

Storing this posting requires 201 bits (26 bytes) and 204 bits (26 bytes) in our implementation, if γ-coding and δ-coding is used, respectively.

3.2 Key Observations

Figure 1 reveals two characteristics that we consider as typical for changes in versioned document collections.

High Level of Redundancy Looking at the word b we see that it occurs at the same positions in document versions d^{t_1} and d^{t_2}. Comparing the two document versions, we see that content was only appended, leaving all existing word positions untouched. Changes that add content to the end of a document are common in practice. Consider, as an example, bulletin boards on the Web where replies to a post are typically presented in chronological order.

Numerical Closeness A second –more subtle– observation can be made when comparing the positions of word c between document versions d^{t_2} and d^{t_3}. Although there is only a single position common between the two document version, we see that word positions are numerically relatively close. Positions $\{8, 9\}$ in the newer document version are just shifted equivalents of $\{6, 7\}$ in the old. When merging the two sequences, as a result of the numerical closeness, the resulting sequence $\{3, 5, 6, 7, 8, 9\}$ has small gaps between its elements. These smaller gaps in turn result in a better compressibility of the merged sequence. Small shifts by a fixed margin are another common kind of change occuring in practice, for instance, when adding missing or removing superfluous words.

3.3 Fusing Multiple Postings

The two observations presented above reveal that we can yield a more compact representation of word positions by constructing postings that cover multiple consecutive document versions. These *fused* postings must preserve all information that was also conveyed in the original postings. We therefore extend the above posting structure to hold word positions for multiple consecutive document versions as follows

$$\langle \ [t_b, t_e) \mid did \mid positions \mid timestamps \mid signatures \ \rangle .$$

The validity time-interval $[t_b, t_e)$ now covers the validity time-intervals of all n covered document versions. Similarly, the third component keeps the sequence of all m positions where the word occurs in any of the document versions. In the fourth component we keep, in increasing order, the timestamps of all but the first document version.

Keeping the timestamp of this first version is not required, because it coincides with the left boundary of the validity time-interval. In the fifth component, for each of the n covered document versions, a bit signature of length m is kept with the i-th bit conveying whether the i-th position is present in this document version.

Again, as for the above original posting structure, there are different alternatives to represent the different components. Our implementation represents validity time-interval boundaries and document identifier explicitly using 64 bit for each. The sequences of positions and timestamps are compressed computing their differences and using either γ-coding and δ-coding. The signatures are again stored explicitly, thus requiring $n \cdot m$ bits in our implementation.

For the problem instance in Figure 1 we obtain the following posting for b when covering all three document versions

$$\langle \ [t_1, now) \mid d \mid 2, 4, 6 \mid t_2, t_3 \mid 110\ 110\ 101 \ \rangle .$$

When using γ-coding or δ-coding storing this posting requires 322 bits (41 bytes) and 291 bits (37 bytes) in our implementation (assuming that timestamps t_i are maintained at the granularity of milliseconds and space one day apart from each other). In contrast, storing three separate postings requires 78 bytes for both coding schemes. Thus, by building the fused posting we save up to 52% of space.

3.4 Query-Processing Overheads

As we argued above, often one can save significant space by building postings that cover multiple consecutive document versions. Depending on how the posting lists are internally organized, these fused postings may affect query processing adversely. To illustrate this, let us assume that we need to retrieve the word positions for a single document version, possibly because an earlier filtering step has pruned all other candidate versions. When we consider a naïve index organization and query processing over it, the full list is read to retrieve the single posting. In this case, we fully profit from the space savings achieved, since less data needs to be read from disk. However, typically high performance search systems employ sophisticated index organizations that could include skip pointers [9] to jump directly (or close) to the desired posting during query processing. In our running example, this sort of index organization means that, in the *worst case* we read the 41 bytes of the fused posting instead of only 26 bytes for the original posting – an overhead factor of 1.58.

4 Optimization Algorithm

Having illustrated the benefits and pitfalls of building postings that cover multiple consecutive document versions, we now describe a method to systematically determine the versions that should be fused together. Our method is tunable by a parameter η specifying the maximally acceptable *worst case* overhead factor for query processing introduced above. It minimizes the total cost required to represent the word positions in the given document versions under the constraint that no overhead factor larger than the parameter η is paid.

For a fixed word w and fixed document d, the method is applied to a sequence of n versions d^{t_i} of the document containing the word. For ease of explanation we assume that the right validity time-interval boundary of the last document version d^{t_n} is given as t_{n+1}.

For the description of the method, we abstract from the concrete posting structure, and let $cost([t_b,\, t_e))$ denote the cost required to represent the posting covering the document versions d^{t_i} with $t_b \leq t_i < t_e$. We make the natural assumption that the cost is monotonous, i.e.,

$$[t_b,\, t_e) \subset [t_{b'},\, t_{e'}) \;\; \Rightarrow cost([t_b,\, t_e)) \leq cost([t_{b'},\, t_{e'}))\ .$$

The method outputs a partitioning $\mathcal{I}$ of the time interval $[t_1,\, t_{n+1})$. For $[t_b,\, t_e) \in \mathcal{I}$ we build a posting covering document versions d^{t_i} with $t_b \leq t_i < t_e$. Formally, the method solves the following optimization problem

$$\begin{array}{l} \underset{\mathcal{I}}{argmin} \sum\limits_{[t_b,\, t_e) \in I} cost([t_b,\, t_e)) \quad \text{s.t.} \\ \forall\, [t_b,\, t_e) \;\in\; I \;:\; valid([t_b,\, t_e)) \quad \text{with} \end{array}$$

$$valid([t_b,\, t_e)) := [t_i,\, t_{i+1}) \subseteq [t_b,\, t_e) \Rightarrow cost([t_b,\, t_e)) \leq \eta \cdot cost([t_i,\, t_{i+1}))\ .$$

Hence, the total cost to represent the word positions in the given document versions is minimized. The family of constraints ensures that we never pay an overhead factor larger than η when processing a query. The predicate $valid([t_b,\, t_e))$ is true only if the cost $cost([t_b,\, t_e)$ of the fused posting is larger by a factor of at most η than any of the original postings' costs.

Computing an Optimal Solution An optimal solution to the above optimization problem is computable efficiently as we explain next. The recurrence

$$\text{OPT}([t_1,\, t_k)) = \underset{1 \leq j < k}{min}\ \{\, \text{OPT}([t_1, t_j)) + cost([t_j, t_k))\ \mid\ valid([t_j, t_k))\ \}\ .$$

allows us to compute an optimal solution for the time interval $[t_1,\, t_k)$ based on the optimal solutions of its prefixes. The optimal solution is determined as the least expensive combination of a prefix $[t_1, t_j)$ and a single valid time interval $[t_j, t_k)$.

Pseudo-code for computing an optimal solution is given in Algorithm 1. First, the algorithm precomputes the costs of original postings $cost([t_i,\, t_{i+1})$ keeping them in cost[i]. Pseudo-code for this precomputation is omitted. Following that, the algorithm computes optimal solutions for increasing prefixes of $[t_1, t_{n+1})$. The optimal cost determined for the prefix $[t_1, t_k)$ is kept in opt[k], the split, i.e., the optimal value j according to the above recurrence is kept in splits[k]. Due to the assumed cost monotonicity, we can terminate the inner loop early, once the validity test fails. The algorithm has time complexity in $O(n^3)$ – in each iteration of the nested loops, the validity of the time interval $[t_j, t_k)$ is checked requiring $O(n)$ comparisons in the worst case. The space complexity is in $O(n)$ for keeping the costs of original postings and prefixes, as well as, the splits. The cubic runtime may seem impractical first, nevertheless we found that the inner loop is most often terminated early for reasonable values of η, making the algorithm well applicable in practice.

```
1  cost [1..n] = ⟨ cost([t_1, t_2)), ..., cost([t_n, t_{n+1})) ⟩;
2  opt [2..n] = ⟨ cost([t_1, t_2)), ∞, ..., ∞ ⟩;
3  splits [2..n] = ⟨ 1, 0, ..., 0 ⟩;
4  for k = 3 to n + 1 do
5      for j = k − 1 to 1 do
6          if valid([t_j, t_k)) then
7              combined = opt[j] + cost([t_j, t_k));
8              if opt[k] > combined then
9                  opt[k] = combined;
10                 splits[k] = j;
11         else
12             break;
```

Algorithm 1: Computing an optimal solution

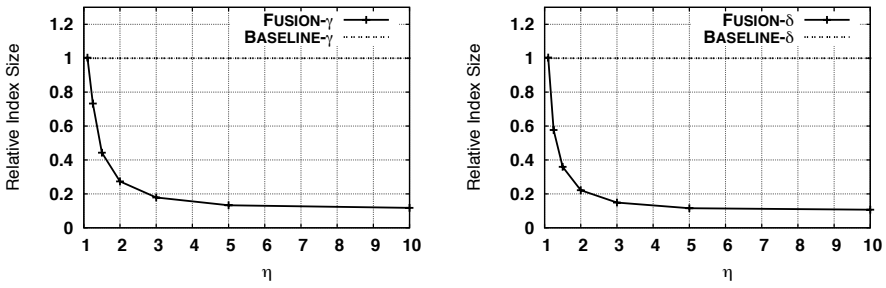
5 Experimental Evaluation

Dataset To evaluate our approach experimentally, we chose the revision history of the English Wikipedia [10] as a dataset. This dataset or parts thereof were also used for the experimental evaluation in [2, 3]. We first extracted all document versions committed during the years 2004 and 2005. From this collection, we extracted all document versions belonging to a randomly sampled 10% subset of the document identifiers. In total, the resulting collection contains $35,309$ documents in $934,411$ versions.

Index Sizes On this collection, we constructed indexes according to the baseline described in Section 3.1 using γ-codes and δ-codes. The resulting indexes are further referred to as BASELINE-γ and BASELINE-δ. In total, their sizes amount to 97.77 GBytes and 97.51 GBytes, respectively. For our method, again using γ-codes and δ-codes, we built indexes varying the threshold η between 1.1 and 10.0. These sets of indexes are referred to as FUSION-γ and FUSION-δ in the remainder.

Figure 2 plots the resulting index sizes relative to the respective BASELINE. It can be seen from the figure that our method is highly effective and achieves a significant reduction of index size even for modest values of η. Thus, as an example, when setting $\eta = 1.5$, which corresponds to a query-processing overhead of at most 50%, the index size is reduced by more than 50% for both coding schemes.

When comparing the effectiveness of our method across the two coding schemes, we observe that it performs slightly better when the more compact δ-coding is employed. For $\eta = 2.0$, as an example, we observe a reduction in index size by close to 80% in Figure 2(b) in contrast to only 75% in Figure 2(a). Hence, there seems to be a reinforcing effect between the effectiveness of our method and the effectiveness of the employed coding scheme.



(a) Elias γ-codes (b) Elias δ-codes

Fig. 2. Relative index sizes for $\eta \in \{1.1,\ 1.25,\ 1.5,\ 2.0, 3.0,\ 5.0,\ 10.0\}$

6 Conclusions

In this work, we have developed a method to compress word positions in a versioned document collection, and thus provided a basis for efficient time-travel phrase querying and proximity-based ranking in these collections. By building postings that cover multiple consecutive document versions, we were able to reduce index size significantly, as demonstrated in our experiments on parts of the revision history of the English Wikipedia.

In our ongoing research, we investigate experimentally how much query processing overhead is imposed by our approach in reality. In the future, we plan to develop query-processing techniques that can leverage fused postings directly to discard entire groups of document versions.

References

1. Peter G. Anick and Rex A. Flynn. Versioning a Full-text Information Retrieval System. In *SIGIR*, 1992.
2. Klaus Berberich, Srikanta Bedathur, Thomas Neumann, and Gerhard Weikum. A Time Machine for Text Search. In *SIGIR*, 2007.
3. Michael Hersovici, Ronny Lempel, and Sivan Yogev. Efficient Indexing of Versioned Document Sequences. In *ECIR*, 2007.
4. Jiangong Zhang and Torsten Suel. Efficient Search in Large Textual Collections with Redundancy. In *WWW*, 2007.
5. Andrei Z. Broder, Nadav Eiron, Marcus Fontoura, Michael Herscovici, Ronny Lempel, John McPherson, Runping Qi, and Eugene J. Shekita. Indexing Shared Content in Information Retrieval Systems. In *EDBT*, 2006.
6. Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers Inc., 1999.
7. Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.
8. Falk Scholer, Hugh E. Williams, John Yiannis, and Justin Zobel. Compression of Inverted Indexes for Fast Query Evaluation. In *SIGIR*, 2002.
9. Alistair Moffat and Justin Zobel. Self-Indexing Inverted Files for Fast Text Retrieval. *ACM Trans. Inf. Syst.*, 14(4):349–379, 1996.
10. http://www.wikipedia.org.

Efficiency Comparison of Document Matching Techniques

Patrice Lacour, Craig Macdonald, and Iadh Ounis

Department of Computing Science,
University of Glasgow, Glasgow, G12 8QQ, UK.
{lacourp,craigm,ounis}@dcs.gla.ac.uk

Abstract. Inverted indices are one of the most commonly used techniques to search very large document collections. While the typical size of web document collections is constantly increasing, users have come to expect a very quick response time, and accurate search results. Hence, to make best use of available hardware resources, efficient and effective retrieval techniques are desirable. We review several state-of-the-art approaches for matching documents to query terms, based on term-centric and document-centric scoring. We test the techniques using three modern Web Information Retrieval (IR) test collections, and conclude in terms of the trade-off between retrieval effectiveness and efficiency.

1 Introduction

Inverted indices are one of the most commonly used techniques to search very large document collections. Usually, these contain for each term a stream of *posting*s, i.e. the id of every document that the term occurs in, and the frequency of the term in that document [18]. Other information may also be stored in the posting list, such as term position information, or the HTML tags where the term occurs within the document [3]. In this paper, we only consider a simple inverted index containing the document identifier and term frequency postings.

The most common method for scoring documents retrieved in response to a query (when using a bag-of-words retrieval approach) is to score each occurrence of a query term in a document using the information contained in its corresponding posting list in the inverted file, and combining these scores for each document. However, for terms with low discriminatory power, then every document the term occurs in must be scored, leading to high retrieval time without benefit to retrieval effectiveness.

While parallelised retrieval can mitigate the cost of high retrieval time, three other matching approaches exist to reduce retrieval times, by trading off with the overall effectiveness of system: Firstly, the pruning of low value documents or terms from the inverted indices [2, 14]; secondly, ordering inverted indices postings by their impact on retrieval [12]; thirdly avoiding scoring all occurrences of the query terms. The focus of this work is to assess various techniques for efficient matching in the latter case. Broadly, these efficient matching techniques

fall into two categories - scoring by each term, or scoring by documents - known as Term-at-a-time (TAAT) and Document-at-a-time (DAAT), respectively.

We experiment using these techniques, and integrate them into an existing IR platform. The platform provides a classical TAAT approach where every term occurrence for each query term is evaluated, and we apply this as the baseline in our experiments. Moreover, several DAAT approaches utilise an improved inverted index format, which allows the decompression of documents that are unlikely to be relevant to be skipped - indeed we experiment with two skipping strategies. All our experiments are conducted in a uniform experimental setting, consisting of several standard Web IR test collections of varying size. We conclude in terms of retrieval effectiveness, retrieval efficiency, and for the inverted index skipping techniques, the additional disk space required by the skipping structures. The contributions of this paper are two-fold: firstly, an in-depth study of the efficiency versus the effectiveness of various matching techniques, and, secondly, to relate the findings to the statistics of the applied test collections. Our work differs from that of other studies such as that of Cambazoglu & Aykanat [7], in that both efficiency *and* effectiveness measures are examined, and trade-off conclusions made.

The remainder of this paper is structured as follows: Section 2 briefly reviews various matching approaches; Section 3 describes several inverted index formats supporting efficient skipping; Section 4 details our experimental setting; and results are provided in Section 5. We provide concluding remarks in Section 6.

2 Matching Techniques

We now describe the matching techniques we experiment with in this paper. As mentioned in the introduction, these fall into two categories - namely TAAT and DAAT. One of the problems with TAAT approaches is that a larger number of documents are scored, requiring more memory to accumulate their scores (known as accumulators [7]). In contrast, DAAT techniques require less memory, as less documents should be scored overall.

However, in all cases except for Full TAAT (the baseline), a trade-off is made by recognising that most Web retrieval tasks favour high-precision retrieval, and aiming to retain high precision effectiveness at the system's top-ranked documents, while degrading retrieval effectiveness at lower ranks. This is typically achieved by predicting which terms have the least impact on the retrieved documents and ignoring portions of their posting lists not likely to affect the top-ranked documents.

Full TAAT. This is our baseline, a TAAT approach, where we compute the score of all postings for every query term. This is in contrast to the other approaches, which aim to avoid some scoring computations, this technique exhibits no degradation in the effectiveness of the ranking.

Turtle TAAT. This technique is a TAAT approach developed by Turtle & Flood in [17]. Firstly, the query terms are ordered by their maximum possible impact on the retrieval score (called the *term upper bounds*). Then, the scoring

of occurrences is performed in two phases: During the first phase, all postings are evaluated for each query term. At the end of the scoring of each query term, if the minimum score of the current top-ranked R documents is greater than the sum of the upper bounds of the query terms remaining to be scored, then the first phase scoring is terminated, and we proceed to the second phase. In this second phase, only the documents which have already been scored are computed for the remaining query terms.

This technique ensures that the top R documents are correctly ordered compared to a Full TAAT, but without the need to fully evaluate all the query terms, leading to document relevance scores that are different from the Full TAAT, but with broadly similar rankings.

Moffat TAAT. The idea for this approach introduced by Moffat and Zobel in [10] is also to score the most important query terms first. However, when K documents have been scored, the matching enters a second phase, where no more documents are retrieved, and the K documents retrieved thus far are fully scored. K is typically equal to 0.2% of the number of documents in the collection. Note that Moffat and Zobel presented two different heuristics, called 'Quit' and 'Continue' to determine when the first phrase scoring should be terminated. In the 'Quit' approach, the condition of accumulators reaching size K is checked for every posting scored, while for the 'Continue' approach, the condition is checked only after each query term has been scored. In this paper, we only experiment using the 'Continue' approach as this has less efficiency overheads, but may result in scoring far more than K documents if a term posting list is very long.

Note also that in the second phase of Turtle TAAT and Moffat TAAT matching approaches, it is possible to omit large numbers of postings for each query term. In this scenario, it is advantageous for the inverted index implementation to support a *next(docid)* operation. In the following section, we review several inverted index skipping techniques.

Lester et al. [9] describe adaptive improvements to the Moffat TAAT approach in the context of Web-style queries. We do not experiment with this approach, as we wish to experiment with generic algorithms without specific assumptions.

Turtle DAAT. This DAAT technique was also developed by Turtle & Flood in [17]. In a DAAT technique, the postings lists for all query terms are read concurrently. Similar to Turtle TAAT, the term upper bounds are computed. Then all of the ith postings for each query term are evaluated in turn. When a document is scored, we add to the actual score of the document, the sum of the upper bound of the remaining terms to be scored. If this sum is less than the minimum score of the current Rth-ranked document, then we do not calculate the term score of the document, as the document could not be in the top R documents.

It is of note that Turtle & Flood [17] suggest that their DAAT technique could be improved by the use of inverted index posting skipping, however it is not clear from their description how this would be applied. Turtle & Flood's methods are known as MAX-SCORE (MAX-SCORE TAAT and DAAT). Moreover,

Stronham et al. [15]. revisit the MAX-SCORE approaches for DAAT, suggesting heuristics to determine the top documents for a given query term, pre-scoring these documents, and using these pre-scored documents as the threshold for the MAX-SCORE algorithm. However, in contrast to the previously described approaches, their approach requires that posting lists be sorted by decreasing tf, not ascending docid.

3 Skipping Inverted Index Postings

Typically query term scoring in an IR system is disk IO intensive, however, for larger Web-scale settings with many machines available, posting lists may be kept in the main system memory or in the system cache [16]. In both cases, compression is useful to reduce IO in favour of slight compression overheads [6]. Some matching methods described above (in particular Moffat TAAT) can potentially avoid scoring some postings of the query terms. Therefore, if this skipping could reduce the IO operations, then the retrieval time can be reduced. Skipping techniques on compressed inverted indices were first described by Moffat & Zobel in [10]. In this work, we review both Moffat & Zobel's approach as well as the later approach proposed by Boldi & Vigna in [3].

Posting compression in inverted indices can be performed in several fashions. One compression technique that can be applied is the use of Elias-Unary and Elias-Gamma encoding [8]. While other techniques such as variable-byte encoding exist (and may decode more efficiently), in this work we concentrate only on these Unary and Gamma encodings.

As a given posting can then be of variable length, to efficiently skip over postings in the posting list, additional pointers have to be embedded in the posting list describing a potential skip when decoding. Unsurprisingly, these additional pointers cause the size of the inverted index to grow.

Assume that the inverted index postings are grouped into blocks (of $p \geq 3$ postings). A pointer allows one or more blocks to be skipped, ensuring that the next posting to be read has a document id greater than or equal to a required document. Skipping is best illustrated following Moffat & Zobel [10]:

In a normal inverted index for a given term, a posting list of ⟨docid,tf⟩ tuples is stored as follows:

$$\langle 5,1\rangle\ \langle 8,1\rangle\ \langle 12,2\rangle\ \langle 13,3\rangle\ \langle 15,1\rangle\langle 18,1\rangle\langle 23,2\rangle\langle 28,1\rangle\langle 29,1\rangle\langle 32,3\rangle$$

We insert the pointers ⟨⟨docid, bit address⟩⟩ every p pointers (say 3 in our example), where the docid in the pointer is the first docid in the first posting of the next block, and a_i is the pointer to the next block. Note that the first pointer is an exception, as a_0 is not required.

$$\langle\langle 5,a_0\rangle\rangle\ \langle\langle 13,a_1\rangle\rangle\ \langle 5,1\rangle\langle 8,1\rangle\langle 12,2\rangle\ \langle\langle 23,a_2\rangle\rangle\ \langle 13,3\rangle\langle 15,1\rangle\langle 18,1\rangle\ \langle\langle 32,a_3\rangle\rangle\ \langle 23,2\rangle\langle 28,1\rangle\langle 29,1\rangle$$

We can compress further the data by removing redundant document ids and by applying delta gaps between pointers.

$$\langle\langle 5,a_0\rangle\rangle\ \langle\langle 8,a_1\rangle\rangle\ \langle 1\rangle\langle 3,1\rangle\langle 1,2\rangle\ \langle\langle 9,a_2-a1\rangle\rangle\ \langle 3\rangle\langle 2,1\rangle\langle 3,1\rangle\ \langle\langle 9,a_3-a_2\rangle\rangle\ \langle 2\rangle\langle 5,1\rangle\langle 1,1\rangle$$

This is a common approach of all of the skipping approaches. It allows us skipping forward by an entire block, without decoding the information encoded in the block.

From [3] & [10], we note three approaches. These differ in the way that they determine the size of a block of postings p. Moreover, two approaches contain multiple levels of pointers, allowing effective skipping of several blocks in one disk seek.

Single Moffat: In this approach, the size of a posting block p for term t is set as follows:

$$\frac{p = \sqrt{LN_t}}{2} \tag{1}$$

where N_t is the number of documents that term t occurs in, and L is a free parameter. Moffat & Zobel suggest three values of L, namely 10, 100 and 1000, depending on statistics of the collection and of the query set [10].

Multi Moffat: In this method, the size of the skip p_i of each level i of term t is determined as a function of the skipping parameter L as follows:

$$p_i = \frac{1}{2} L^{\frac{i}{h+1}} p^{\frac{h-i+1}{h+1}} \tag{2}$$

where h is the number of levels to be computed for this term, calculated as:

$$h = (log_e \frac{N_t}{L}) - 1 \tag{3}$$

Multi Boldi: The lowest level has a block of L (32 or 64) postings, and each higher level is composed of 2 lower ones [3]. Moreover, when a posting list does not have exactly an exact multiple of L postings, extra unconnected pointers can exist. In this approach they are removed, to prevent posting lists being unnecessary large.

In the following section, we experiment with the Single Moffat and Multi Boldi skipping techniques, as these are two representatives of the single and multiple skipping level classes of techniques. Experiments using Multi Moffat remain as future work.

4 Experimental Setting

In the following, we experiment with the reviewed matching techniques and inverted index skipping techniques. To test the matching methods, we use three different Web IR test collections, related to different domains and timescales. The collections we experiment with are: two TREC Web test collections WT2G and WT10G, which are medium-scale general Web crawls from early 1997; .GOV2 is a more recent and much larger TREC Web test collection - a crawl of the .gov domain of the Web from 2003. Statistics from the collections are given in Table 1, as well as the TREC topic sets applied. Note that, as expected, the average length of the term postings increases as the collection size grows. This infers that for larger collections, there is more potential improvements for

Collection	#Docs	Avg Doc Len	#Terms	Avg Posting Len	Topics
WT2G	247,491	645.3	1,002,586	62.7	401-450
WT10G	1,692,096	399	3,140,838	89.1	451-500
GOV2	25,205,179	652.4	15,466,363	304.4	801-850

Table 1. Statistics of the Web IR test collections applied.

increasing the retrieval speed when skipping can be used. Finally, we experiment with short (title-only) queries and long (title, description and narrative) queries from the used test collections, to assess whether the topic length has an impact on the efficiency or effectiveness retrieval performances[1].

In most retrieval tasks, the user is interested in the accuracy of the first page of documents retrieved. For this reason, in all methods, we set R, the number of top-ranked documents which should be fully scored, to $R = 20$.

We use the Terrier IR platform [11] in our experiments. The collections are indexed, removing standard English stopwords, and applying Porter's English stemmer. For retrieval we use the BM11 document weighting model (as this is similar to the TF-IDF which was applied in the original papers on DAAT and TAAT matching approaches) [13], as follows:

$$score(d, Q) = \sum_{t \in Q} qtw \cdot \frac{tf \cdot k_1}{tf + k_1 \cdot (\frac{1-b+b \cdot l}{avg\_l})} \cdot log_2 \frac{N}{\frac{N}{N_t+1}} \quad (4)$$

where tf is the frequency of query term t in document d, l is the length of document d, avg_l is the average length of all documents in the collection, N is the number of documents in the collection, and N_t is the number of documents containing an occurrence of term t. k_1 and b are parameters, for which we use the default settings $k = 1.2$ and $b = 0.75$.

Note that BM11 includes a document length normalisation component, to ensure a fair retrieval between long and short documents. By applying normalisation, the frequency of a term tf is normalised by the length of the document l, with respect to the average document length avg_l. However, this normalisation can possibly hinder the correct calculation of the maximum score a term in a document can achieve (the term upper bound), depending on whether the posting with largest tf also has smallest document length. To simplify, we approximate document length with average document length in the collection when calculating term upper bounds using BM11, as this avoids pre-scoring every posting for every term in the collection, in order to obtain exact upper bounds. However, since tf can exhibit strong correlations with l [1], average document length may not be a good approximation. A more accurate approximation of the term upper bounds is an appropriate future work direction.

For experiments where timing is recorded, these were carried out using a dedicated Intel PIV 2GHz single CPU machine, with 512MB of RAM, running Linux. The file-store is a RAID array attached to a Linux server by 20MB/sec Wide SCSI. Machines are inter-connected by 100MBPs network.

[1] Long queries on GOV2 are omitted.

Matching	Short			Long		
	#Term Scorings	P@20	MAP	#Term Scorings	P@20	MAP
WT2G						
Full TAAT	24,926	0.3650	0.2613	498,966	0.3620	0.2784
Turtle TAAT	7,202	0.2411	0.1788	253,991	0.1980	0.1262
Turtle DAAT	23,649	0.3230	0.1300	253,911	0.0350	0.0059
Moffat TAAT	6,855	0.3460	0.2350	16,142	0.1560	0.0857
WT10G						
Full TAAT	183,607	0.2030	0.1880	2,039,356	0.2810	0.2326
Turtle TAAT	29,553	0.1740	0.1295	740,658	0.1440	0.1043
Turtle DAAT	163,371	0.1690	0.1054	1,253,580	0.0290	0.0090
Moffat TAAT	27,845	0.1870	0.1793	49,860	0.1530	0.0803
GOV2						
Full TAAT	2,221,422	0.2840	0.2244	-	-	-
Turtle TAAT	692,921	0.2360	0.1402	-	-	-
Turtle DAAT	2,003,572	0.2190	0.1134	-	-	-
Moffat TAAT	628,538	0.2330	0.1320	-	-	-

Table 2. Efficiency and effectiveness results applying various matching techniques to the three Web test collections.

5 Experimental Results

Table 2 presents the experimental results for the tested matching approaches. Effectiveness is measured using Mean Average Precision (MAP) and Precision at rank 20 (P@20). P@20 is a useful measure as it represents the accuracy of the first screen of search results presented to a search engine user [5]. Efficiency is measured by the average number of query term-document scorings per query. The results presented for all methods do not use any additional inverted index structure for skipping postings, instead emulating skipping by simply iterating through the postings until the required document is found.

From the result in Table 2, we note firstly that all three 'efficient techniques' (i.e. Turtle TAAT, Turtle DAAT and Moffat TAAT) can reduce the number of term occurrence scorings compared to the Full TAAT (our baseline matching technique). However, such increase in efficiency has a corresponding trade-off in retrieval effectiveness - this is typically more marked for MAP than P@20. This is expected, as all the efficient techniques aim to get the top ranked documents correct while trading off overall retrieval performance.

Comparing the techniques for short queries, it is noticeable that Moffat TAAT matching usually achieves the lowest overall drop in P@20 and MAP. This is surprising, given that far less term occurrences are scored when compared to Turtle DAAT, which has slightly lower MAP and P@20 scores. Turtle TAAT is more comparable to Moffat TAAT in terms of number of scorings, but less so in terms of retrieval performance.

Examining the long queries, it is noticeable that all of the efficient matching techniques perform less well here than on short queries. This contrasts from the baseline matching technique, which usually increases in effectiveness (MAP and

P@20). In particular, Turtle DAAT exhibits very low effectiveness. We believe that the low performance of the efficient techniques for long queries is because each approach tries to ascertain the impact of each query term in the retrieval process. Indeed, for Turtle DAAT, the order of term scoring may be predicted incorrectly, and hence the more important query terms occurring in the title of the topics may not be scored. It is of note that the collection originally tested by Turtle & Flood was much smaller than WT2G and WT10G (11K docs vs. 200K docs and 1.6 million docs respectively), and only medium length queries (on average, 9.1 terms each) were used [17].

Comparing techniques across collections, it is noticeable that the efficiency benefit of applying efficient techniques grows for larger collections, as more term occurrences can be omitted from the scoring. Indeed, for GOV2 with short queries, Moffat TAAT requires only 72% of term occurrences scorings compared to the Full TAAT baseline for a 17% drop in P@20.

Overall, as expected, the techniques reviewed here can be applied to speed up retrieval, if some degradations in retrieval effectiveness can be tolerated. Of all the techniques, Moffat appears the most promising in reducing term occurrence scorings while minimising the corresponding reduction in retrieval performance.

Table 3 presents a comparison of the time taken to perform retrieval for short and long queries on the the WT2G and WT10G collections using the Turtle TAAT and Moffat TAAT approaches. Using these efficient matching approaches, we compare the efficiency of two inverted index skipping approaches reviewed in Section 3. The baseline inverted index does not skip any postings.

From Table 3, we firstly note that (unsurprisingly) index sizes are increased by the inclusion of skipping pointers to the inverted index. For some collections, this can be as high as a 46% increase (see WT2G, Single Moffat $L = 1000$). However, as collection size increases, the overhead in index size associated with the skipping pointers decreases (down to 23% increase for $L = 1000$ on WT10G).

In terms of retrieval time, we found that most inverted index skipping settings improved the average query time, particularly on WT2G. On this collection, the retrieval efficiency is benefited most by Moffat TAAT, particularly with Single Moffat skipping. For WT10G, there is no large benefits in applying skipping techniques, and the most benefit is shown when using Multi Boldi (with minimum skip size $L = 32$) for short queries. Comparing short and long queries in general, the differences in retrieval times are largely correlated with the number of term occurrence scorings reported in Table 2 above. Overall, we conclude that the application of the skipping methods only benefited retrieval time on WT2G. For WT10G, it appears that the overheads in decoding the more complicated inverted index format outweighs the benefit in reducing the disk IO by skipping.

While the overall timing statistics may change if other compression techniques (e.g. variable-byte encoding) had been applied, the conclusions would not change, as the compression methods are not varied and the comparisons in Table 3 are fair.

	No Skipping	Single Moffat			Multi Boldi	
$L =$	-	10	100	1000	32	64
WT2G						
SQ Turtle TAAT	0.0899	0.0804	0.0803	0.0806	0.0804	0.0803
LQ Turtle TAAT	0.549	0.438	0.440	0.438	0.612	0.438
SQ Moffat TAAT	0.0962	0.0829	0.0831	0.0840	0.0841	0.0841
LQ Moffat TAAT	0.202	0.154	0.159	0.187	0.222	0.176
Index Size	100%	122%	136%	146%	110%	105.8%
WT10G						
SQ Turtle TAAT	0.304	0.312	0.315	0.307	0.277	0.409
LQ Turtle TAAT	2.271	2.227	2.244	2.314	2.275	2.240
SQ Moffat TAAT	0.304	0.335	0.327	0.342	0.293	0.319
LQ Moffat TAAT	0.691	0.819	0.839	0.899	0.879	0.800
Index Size	100%	111%	115%	123%	107.8%	104.3%

Table 3. Mean query time (seconds) using the Moffat TAAT and Turtle TAAT matching techniques for the various collections and inverted index skipping techniques. SQ and LQ denote short and long queries, respectively. Index sizes are also reported; L is the inverted index skipping parameter. Experiments on GOV2 are omitted.

6 Conclusions

In this paper, we reviewed several approaches for efficiently matching and scoring documents in response to queries, and evaluated their efficiency and effectiveness on three standard Web IR test collections. We found that, compared to a full TAAT scoring baseline, these approaches could markedly reduce the number of term occurrence scorings, but at the cost of reduced overall effectiveness (MAP). However, high precision accuracy was usually maintained. There is anecdotal evidence in this paper that the efficient matching techniques can give larger benefits in retrieval performance for larger collections. However, improvements in applying inverted index skipping techniques did not appear to scale to a larger collection. In the future, we intend to investigate how efficiency changes when using larger test collections such as UK-2006, and comparing to the systems submitted to the Efficiency task in the TREC 2004-2006 Terabyte tracks [5]. In particular, it is probably more useful to evaluate effectiveness using 50 assessed queries, but evaluating efficiency using larger number of queries, to gain more accurate timing measurements, as performed in the TREC Terabyte track Efficiency tasks.

Other future investigations will include study into other techniques for reducing the number of term occurrences scored, such as the newer versions of Turtle TAAT [9], MAX-SCORE DAAT [15], and finally the WAND iterator proposed by Broder et al. in [4]. We would also like to understand more fully the relationship between retrieval performance and the R parameter (which controls the number of documents that are scored and retrieved), especially with respect to the application of document priors commonly required for Web search tasks such as home-page and named-page finding tasks.

References

1. Amati G.: *Probabilistic Models for Information Retrieval based on Divergence from Randomness*. PhD thesis, Department of Computing Science, University of Glasgow, 2003.
2. Blanco R., Barreiro A.: Static Pruning of Terms in Inverted Files. In *Proceedings of ECIR 2007*: (2007) 64–75
3. Boldi, P., Vigna, S.: Compressed Perfect Embedded Skip Lists for Quick Inverted-Index Lookups. In *Proceedings of SPIRE 2005* LNCS 3772 (2005) 25–28
4. Broder A., Carmel D., Herscovici M., Soffer A., Zien J.: Efficient Query Evaluation using a Two-Level Retrieval Process. In *Proceedings of CIKM 2003* (2003) 426–434
5. Buttcher S., Clarke C.L.A., Soboroff I.: The TREC 2006 Terabyte Track. In *Proceedings of TREC 2006*, (2007)
6. Buttcher S., Clarke C.L.A.: Index Compression is Good, Especially for Random Access. In *Proceedings CIKM 2007*, (2007)
7. Cambazoglu B. B., Aykanat C.: Performance of query processing implementations in ranking-based text retrieval systems using inverted indices. *Inf. Process. Manage.* **42**(4) (2006) 875–898
8. Elias, P.: Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory* **21**(2) (1975) 194–203
9. Lester N., Moffat A., Webber W., Zobel J.: Space-limited ranked query evaluation using adaptive pruning. In *Proceedings of the WISE Workshop on Information Systems Engineering* (2005) 470–482
10. Moffat A., Zobel J.: Self Indexing Files for Fast Text Retrieval *ACM Transactions on Information Systems* **14**(4) (1996) 349–379
11. Ounis I., Amati G., Plachouras V., He B., Macdonald C., and Lioma C.: Terrier: A high performance and scalable information retrieval platform. In *Proceedings of SIGIR OSIR Workshop 2006* (2006)
12. Persin M., Zobel J., Sacks-Davis R.: Filtered document retrieval with frequency-sorted indexes. *J. American Society of Information Science*, **47** (1996)
13. Robertson S.E., Walker S., Jones S., Hancock-Beaulieu M.M., Gatford M.: Okapi at TREC-3 In *Proceedings of TREC 3* (1994)
14. Soffer A., Carmel D., Cohen D., Fagin R., Farchi E., Herscovici M., Maarek Y.S.: Static Index Pruning for Information Retrieval Systems. *Proceedings of ACM SIGIR 2001* (2001) 43–50
15. Strohman T., Turtle H., Croft W.B.: Optimization strategies for complex queries In *Proceedings of ACM SIGIR 2005* (2005) 219-225
16. Strohman T., Croft W.B.: Efficient document retrieval in main memory In Proceedings of ACM SIGIR 2007 (2007) 175–182
17. Turtle H., Flood J.: Query Evaluation : Strategies and Optimisations. *Information Processing & Management*: **31**(6) (1995) 831-850
18. Witten I.H., Moffat A., Bell T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images Morgan Kaufmann (1999).

Evaluation of a M-Tree in a Content-Based Image Retrieval System

David García-Pérez[1], Antonio Mosquera[1], Stefano Berretti[2], and Alberto del Bimbo[2]

[1] Departamento de Electrónica y Computación,
Universidade de Santiago de Compostela, Spain
davidgp@usc.es, mosquera@dec.usc.es
[2] Dipartimento di Sistemi e Informatica,
University of Firenze, Italy
{berretti, delbimbo}@dsi.unifi.it

Abstract. After validating an image feature extraction system (an Active Net) and created a metric for the extracted features (casting the Active Nets to graphs), it is necessary to make an index structure that speeds up the search process [1, 2], reducing the necessary I/O to disk. The M-Tree, due to its properties was selected as an indexing structure. The high dimensionality of the extracted features and its non-vector structure made the M-Tree one of the logical choices. The high dimensionality of the graphs and the non multidimensional vector structure of them made the M-Tree the logical choice [3].

1 Introduction

Effective access to modern archives of digital images requires that conventional searching techniques based on textual keywords are extended by content-based queries addressing visual features of searched data. To this end, many solutions have been experimented which permit to represent and compare images in terms of quantitative indexes of visual features. In particular, different techniques have been identified and experimented to represent content of single images according to low-level features, such as color, texture, shape and structure, intermediate-level features of saliency and spatial relationships, or high-level traits modeling the semantics of image content [4–7].

Among these approaches, image representations based on chromatic indexes have been largely used for general purpose image retrieval systems, as well as for object based search partially robust to changes in objects shape and pose. Several low level features have been considered. In particular, representations based on chromatic indexes have been widely experimented and comprise the basic backbone of most commercial and research retrieval engines such as QBIC [8], Virage [9], Visual Seek [10] or Simplicity [11]. Together with color, texture is a powerful discriminating feature, present almost everywhere in nature. Textures may be described according to their spatial, frequency or perceptual properties.

Features of the appearing shape of imaged objects have also been used to represent image content through a variety of approaches. Only few approaches have tried to conjugate color and shape information to improve the significance of object representations.

We proposed [1, 2] a descriptor modeled through a graph which accounts for structural elements and color of regions (objects) of interest in an image. The graph directly corresponds to an elastic structure (active net) that, through a deformation process, is used to separate regions from the background. In particular, due to their deformable structure, active nets can adapt to the borders and internal part of a region encoding, at the same time, information of color, shape and spatial structure of the region. Now, to improve the efficiency of the search process, the graph representations are stored into an efficient secondary memory access structure using M-Tree indexing. The performance of the indexing retrieval system are evaluated based on a large and publicly available image object database.

The paper is organized in the following sections: In Sect.2, active nets are defined and feature extraction based on active nets is discussed; In Sect.3, a graph based modeling of the active nets is proposed, and a similarity measure between active nets is defined; Efficient image indexing of active nets is proposed in Sect.4; An experimental evaluation of the efficiency and effectiveness of the indexing system is reported in Sect.5; Finally, in Sect.6 discussion and summary of the chapter are given.

2 Modeling image content by active nets

An active net is a discrete implementation of an elastic sheet [12]. In parametric form, it can be defined as $u(r,s) = (x(r,s), y(r,s))$, where $(r,s) \in ([0,1] \times [0,1])$. The domain of parameters (r,s), is discretized to a regular grid of *nodes* defined by the internode spacing. The two-dimensional net can deform in the image space under the control of the following energy function

$$E(u) = \sum_{(r,s)} \left(E_{int}(u(r,s)) + E_{ext}(u(r,s)) \right) \tag{1}$$

The internal energy, E_{int}, controls the shape and structure of the net and is defined as $E_{int}(u) = \alpha \left(|u_r|^2 + |u_s|^2 \right) + \beta \left(|u_{rr}|^2 + 2|u_{rs}|^2 + |u_{ss}|^2 \right)$, where the subscripts indicate the partial derivatives and α and β are coefficients controlling the first and second order smoothness of the net. In particular, the first derivatives make the net contract and the second derivatives enforce smoothness and rigidity of the net.

The external energy, E_{ext}, accounts for external forces acting on the net, and is defined as $E_{ext}(u) = f[I(u)]$, where f is a general function of the properties of the image $I(u)$. The objective is to find a function f that makes the nodes of the net to be attracted to significant zones of an image according to an energy minimization process (*fitting*) [12].

Nodes are divided into *external* nodes (i.e., nodes belonging to the border of the net $u(r,s):\ r,s=0,1$), and *internal* nodes (i.e., nodes that do not belong to the border of the net $u(r,s):\ r,s\neq 0,1$). In our approach, color information is used to drive the fitting process which controls the deformation of a net. This is obtained by creating two external energy images, one locating the borders of a relevant region, and the other describing its internal area. During the net fitting process, the external nodes are attracted to the borders of a region, while the internal nodes are attracted to the internal zones of a region [1, 2].

3 Similarity model for topological active nets

In the perspective to compare objects for retrieval purposes, a distance measure between two active nets has been defined [1, 2]. To this end, a net is regarded as an attributed relational graph so that graph properties can be usefully exploited to enhance the net representation. In this way, the comparison between two nets is reduced to the problem of matching their corresponding graphs.

Given an active net $u\,(r,s)$, it is cast to a graph G by mapping nodes of the net to vertices of the graph, and links between nodes of the net to edges of the graph:

$$\begin{aligned} G &= < V, E, \gamma, \delta >, \\ & V = set\ \ of\ \ vertices \\ & E \subseteq V \times V = set\ \ of\ \ edges \\ & \gamma : V \mapsto L_V,\ \ vertices\ \ labeling\ \ function \\ & \delta : E \mapsto L_E,\ \ edge\ \ labeling\ \ function \end{aligned}$$

where L_V and L_E are the sets of vertices and edge labels, respectively.

In our framework, active nets adapt to image regions according to the overall energy function of Eq.(1) so that nodes are constrained to some relevant point of the image (those providing the stable minimal configuration for the energy of the net). In so doing, the average color (in the $L^*a^*b^*$ color space) of the Voronoi regions surrounding the nodes in the image is used as the vertices labeling function γ of the graph. Given two vertices v_1 and v_2, their distance is evaluated as: $\mathcal{D}_v(v_1,v_2) = \sqrt{(L^*_{v_1}-L^*_{v_2})^2+(a^*_{v_1}-a^*_{v_2})^2+(b^*_{v_1}-b^*_{v_2})^2}$.

In order to account for the deformation of the net with respect to its initial configuration, the normalized distance existing between two nodes n_1 and n_2 is used as edge labeling function δ of the edge e_{n_1,n_2} The distance between edges e_j and e_k is defined as: $\mathcal{D}_e(e_j,e_k) = |l_{e_j} - l_{e_k}|$ being l_{e_j} and l_{e_k} the labels associated to e_j and e_k, and measuring their length.

The comparison of the graph models of a *query net* Q and an archive *description net* D involves the association of the vertices in the query with a subset of the vertices in the description. Using an additive composition, and indicating with $\Gamma(v_i)$ an injective function which associates vertices v_i in the query graph with a subset of the vertices in the description graph, this is expressed as follows:

$$\begin{aligned} \mu^{\Gamma}(Q,D) =& \lambda \sum_{k=1}^{N_q} \mathcal{D}_v(v_k, \Gamma(v_k)) + \\ & (1-\lambda) \sum_{k=1}^{N_q} \sum_{h \in C(k)} \mathcal{D}_e([v_k,v_h],[\Gamma(v_k),\Gamma(v_h)]) \end{aligned} \tag{2}$$

where N_q is the number of vertices in the query graph Q, $C(k)$ is the set of query graph vertices directly connected to the vertex v_k, and $\lambda \in [0,1]$ balances the mutual relevance of edge and vertex distance (e.g., for $\lambda = 1$, the distance only accounts for the chromatic distance).

We assume that nets with the same number of nodes are used to describe every object in the image database This is motivated by the fact that nets with the same number of nodes represent image objects at the same spatial resolution. A second basic assumption is that during comparison, only homologous graph vertices can match This corresponds to assume that the injective function Γ of Eq.(2) maps any vertex v_k in the graph Q to the homologous vertex d_k in D. In so doing, indicating with T_v and T_e the time spent in computing the vertex and the edge distance, respectively, the complexity in matching two graphs is upper bounded by $O(mn \cdot T_v + 4mn \cdot T_e)$ from an exponential time [13].

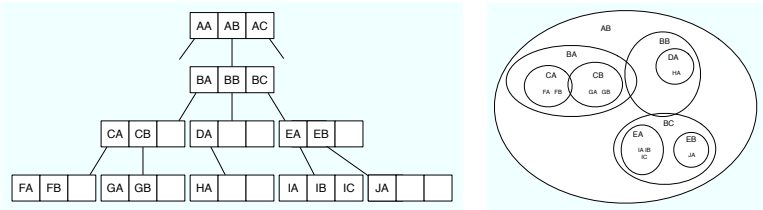
4 Indexing Graphs: M-Tree

The selection of M-Tree [14] for tasks of indexing is motivated by the fact that the graph model proposed in this work is not a feature vector space, so it is not possible to use the traditional tree structures for its indexing. The M-Tree is a paged metric tree. It is a balanced tree, able to deal with dynamic data fill, and it does not require periodical reorganizations. M-Tree can index objects using features compared by distances functions which either do not fit into a vector space or do not use a L_p metric. Using a specific distance function d, the M-Tree can partition objects, and store these objects into fixed nodes, which correspond to constrained regions in the metric space. The only requirement is that the distance follows the metric axioms. If the metric assumption is true, the M-Tree can index any kind of data without knowing their structure.

A M-Tree of graphs is a tree of *nodes*, each containing a fixed maximum number m of *entries*. In turn, each entry is constituted by a routing graph D; a reference to the root sub^D of a (sub)index containing the graphs in the so-called *covering region* of D; and a radius μ^D providing an upper bound for the distance between D and any graph in its covering region (figure 1)

$$\begin{array}{c} < node > ::= \{< entry >\}^m \\ < entry > ::= D, sub^D, \mu^{rD} \end{array} \tag{3}$$

The index tree can be constructed using different schemes for the insertion of new graphs and the selection of routing graphs [14]. In our particular case, the index is constructed dynamically by inserting graphs from the bottom layer and by splitting nodes and promoting routing graphs when insertion overflows occur (see figure 2). In so doing, the tree is kept balanced while its depth grows through splits of the root node. Different policies can be implemented to select the most suitable leaf node for the insertion, the entries moved in the split of a node, and the graphs which is promoted in the split. Typical policies are *Minimum*



(a) An M-Tree which a maximum of 3 objects per node.

(b) Covering radius to each one of those routing objects.

Fig. 1. Visual concept of the covering radius to each one of these routing objects.

of Maximum of Radii (which reduces the size of regions) and *Maximum Lower Bound on Distance* (which reduces the overlap between different regions) [14].

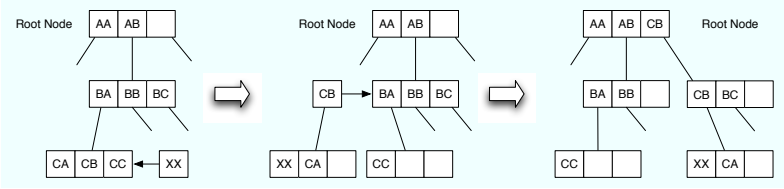


Fig. 2. Split process of the M-tree. The new object XX needs to be inserted into a full leaf node: this induces a split of the node. Two new routing objects are selected, and the leaf node splits into two different leaf nodes. The new routing objects create a new split into a routing node. The split process propagates towards the root of the tree and ends up when no more splits are needed..

In our case, we used the *Minimum Sum of Radii* policy. This algorithm considers all the possible pairs of objects, partitions the set of entries to each one of those combinations and promotes the pair of objects for which the sum of covering radius, $\mu^{D_1} + \mu^{D_2}$ is minimum. This algorithm is the most complex in terms of distance computations.

During retrieval, triangular inequality can be used to support efficient processing of range queries, i.e., queries seeking for all the graphs in the archive which are within a given range of distance from a query graph Q. To this end, the distance between Q and any graph in the covering region of a routing graph D can be lower-bounded using the radius μ^D and the distance between D and Q. Specifically, if μ_{max} is the range of the query, the following condition can be employed to check whether all the graphs in the covering region of D can be discarded, based on the sole evaluation of the distance $\mu(Q, D)$

$$\mu(Q, D) \geq \mu_{max} + \mu^D \rightarrow \textit{ no graph in } sub^D \textit{ is acceptable} \tag{4}$$

In a similar manner, the following condition checks whether all the graphs in the covering region of D fall within the range of the query (in this case, all the graphs in the region can be accepted)

$$\mu(Q, D) \leq \mu_{max} - \mu^D \rightarrow \textit{ every graph in } sub^D \textit{ is aceptable} \tag{5}$$

In the critical case that neither one of the previous two inequalities holds, the covering region of D may contain both acceptable and non acceptable graphs, and the search must be repeated on the subindex sub^D.

K-Nearest Neighbor (KNN) queries seeking for the K graphs that are most similar to the query in the archive can be also managed in a similar manner, but with lower efficiency. This is obtained by regarding the query as a particular case in which the range is determined during the search (This is the solution implemented in this work). To manage KNN queries, Ciaccia et al. [14] proposed the use of a branch-and-bound technique. This technique uses two global structures, a priority queue PR, and a result array NN.

PR is a structure that contains pointers to every subtree that may contain valid objects for a given query Q. For every pointer, a lower bound in the distance of any graph in sub^{D_r} is kept

$$d_{min}\left(sub^{D_r}\right) = max\left\{d(D_r, Q) - \mu^{D_r}, 0\right\} \tag{6}$$

At each step of the algorithm, the entry in PR with the lower d_{min} is considered in order to find objects relevant to the given query.

NN is the sorted array that at the end of the execution contains the final results of the search. Each element of the array stores a graph D_j and its distance to the query, $d(D_j, Q)$. We denote the distance d_k as the distance of the last element in the array. As a consequence, any subtree so that $d_{min}\left(sub^{D_r}\right) > d_k$ can be safely pruned from the search. In so doing, d_k plays the role of a dynamic search radius.

Apart from the lower bound, an upper bound is also considered

$$d_{min}\left(sub^{D_r}\right) = max\left\{d(D_r, Q) - \mu^{D_r}, 0\right\} \tag{7}$$

After analyzing the root graph of a branch, if $d_{max} < d_k$, this root graph is added to NN with d_{max} as distance value, and d_k is updated. This may cause a pruning in PR, thus reducing the number of necessary computations.

5 Results

To evaluate the M-Tree indexing algorithm we used the ETH-80 image database [15]. This database is composed of 3280 images. After being processed by our algorithm the final database was composed of 7016 objects, usually one active net for the object in the image and another for the background. To do a linear scan in this database, comparing one query object against all the objects in it, takes 7016 comparisons and about 347646 miliseconds, using an active net of size

15×15. This time includes I/O access[3]. It is necessary to note that the graphs of size 15×15 have a high consume in memory. This increases the use of the memory, and it makes necessary to do more I/O tasks, increasing the necessary time to complete a search process.

To check the influence of the size of leafs per branch we constructed several trees of different size: 10, 15, 20 and 30 leafs per branch. Every tree was tested with the same query images and compared their results against each other and against a linear scan. The result are resumed in the table 1 (Increasing the size of the leafs per branch over 30 did not show an increase in the performance).

	Time (ms)	**# Comparisons**	**Performance Gain vs Linear Scan (Time)**	**Performance Gain vs Linear Scan (Comparisons)**
10 Leafs per Branch				
Query 1	206264	2337	1.68	3.00
Query 2	119894	1279	2.89	5.48
Query 3	259289	3058	1.34	2.29
Query 4	81569	903	4.26	7.77
15 Leafs per Branch				
Query 1	198023	2266	1.68	3.10
Query 2	136621	1336	2.89	5.25
Query 3	261834	3124	1.32	2.24
Query 4	84047	899	4.13	7.80
20 Leafs per Branch				
Query 1	182913	2156	1.90	3.25
Query 2	126806	1355	2.74	4.21
Query 3	256015	3042	1.35	2.31
Query 4	85438	951	4.06	7.37
30 Leafs per Branch				
Query 1	211128	2274	1.64	3.08
Query 2	154981	1666	2.24	4.21
Query 3	280959	2314	1.23	3.03
Query 4	104521	1197	3.32	5.86

Table 1. Time and number of comparisons for different queries using M-Trees with 10, 15, 20 and 30 leafs at maximum per branch.

6 Conclusions

In this work, we have propose an index system that completes the necessary steps for an already proposed and validated content-based image retrieval system [1, 2]. The metric similarity measure used to compare two graphs allows to index

[3] The tests were done in an Intel Core 2 Duo machine at 2.4GHz, 2 GB of RAM and Java 6.0 Virtual Machine.

all the objects in the database into a M-Tree indexing structure. This speeds up the retrieval process making necessary less computations with respect to a linear scan. Experimental results validated the proposed approach on a standard object image database.

References

1. García-Pérez, D., Mosquera, A., Berretti, S., del Bimbo, A.: Object-based image retrieval using active nets. In: Proceedings International Conference on Pattern Recognition (ICPR'06), Hong-Kong, China (2006) pp. 750–753
2. García-Pérez, D., Mosquera, A., Berretti, S., del Bimbo, A.: Topological active-nets for object-based image retrieval. In: International Conference on Image Analysis and Recognition. (2006) 636–647
3. Böhm, C., Berchtold, S., Keim, D.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys (CSUR) **vol. 33**(n. 3) (2001) pp. 322–373
4. Gupta, A., Jain, R.: Visual information retrieval. Communications of the ACM **40**(5) (1997) pp. 70–79
5. del Bimbo, A.: Visual Information Retrieval. Morgan Kaufmann Plublisers, Inc. (1999)
6. Smeulders, A., Worring, M., Santini, S., Gupta, A., Jain, R.: Content-based image retrieval at the end of the early years. IEEE Transactions on Pattern Analysis and Machine Intelligence **vol. 22**(n. 12) (2000) pp. 1349–1380
7. Lew, M., Sebe, N., Djereba, C., Jain, R.: Content-based multimedia information retrieval: State of the art and challenges. ACM Transactions on Multimedia Computing, Communications and Applications **2**(1) (2006) pp. 1–19
8. Flickner, M., Swahney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D., Yanker, P.: Query by image and video content: The qbic system. IEEE Computer **vol. 28**(n. 9) (September 1995) pp. 23–32
9. Swain, M., Ballard, D.: Colour indexing. International Journal of Computer Vision **vol. 1**(n. 7) (1991) pp. 11–32
10. Smith, J., Chang, S.: Visualseek: a fully automated content-based image query system. ACM Multimedia (1996)
11. Wang, J., Li, J., Wiederhold, G.: Simplicity: Semantics-sensitive integrated matching for picture libraries. IEEE Transactions on Pattern Analysis and Machine Intelligence **vol. 23**(n. 9) (2001) pp. 947–963
12. Ansia, F., Penedo, M., Mariño, C., López, J., Mosquera, A.: Automatic 3d shape reconstruction of bones using active net based segmentation. In: 15th International Conference on Pattern Recognition, Bacelona. (2000)
13. Berretti, S., del Bimbo, A., Vicario, E.: Efficient matching and indexing of graph models in content-based retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence **vol. 23**(n. 10) (2001) pp. 1089–1105
14. Ciaccia, P., Patellla, M., Zezula, P.: M-tree: An effciente access method for similarity search in metric spaces. In: 23rd VLDB Conference. (1997)
15. Leibe, B., Schiele, B.: Analyzing appearance and contour based methods of object categorization. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'03). (2003)

Metric inverted - an efficient inverted indexing method for metric spaces *

Benjamin Sznajder, Jonathan Mamou, Yosi Mass, and Michal Shmueli-Scheuer

IBM Haifa Research Laboratory, Mount Carmel Campus, Haifa, Israel
{benjams,mamou,yosimass,shmueli}@il.ibm.com

Abstract. The increasing amount of digital audio-visual content accessible today calls for scalable solutions for content based search. The state of the art solutions reveal linear scalability in respect to the collection size due to the large number of distance computations needed for comparing low level audio-visual features. As a result, search in large audio-visual collections is limited to associated metadata only done by text retrieval methods that are proven to be scalable.
Search in audio-visual content can be generalized to search in metric spaces by assuming some distance function on low-level features. In this paper we propose a framework for efficient indexing and retrieval of metric spaces by extending classical techniques taken from the textual IR methods such as lexicon, posting lists and boolean constraints- thus enable scalable search in any metric space and in particular in audio-visual content. We show the efficiency and effectiveness of our method by experiments on a large image collection.

1 Introduction

Recently, with the proliferation of Web 2.0 applications, we experience a new trend in multimedia production where users become mass producers of audio-visual content. Websites like Flickr, YouTube and Facebook[1] are just some examples of websites with large collections of audio-visual content. Still, search in such collections is limited to associated metadata that was added manually by users and thus search results are quite dependant on the quality of the metadata.

The problem of audio-visual content based search can be generalized to a search in metric spaces by assuming some distance function on low-level features. For example, for CBIR (Content Based Image Retrieval) it is common to use low level features such as Scalable Color, Textures, Edge Histograms etc. These features define, each one, a specific metric space [7].

State of the art solutions for search in metric spaces such as [2, 3, 10] reveal linear scalability in the collection size. This is mainly because of high number of distance computations needed for comparing low level features. Thus, some

* This work was partially funded by the European Commission Sixth Framework Programme project SAPIR - Search in Audiovisual content using P2P IR.

[1] http://www.flickr.com, http://www.youtube.com, http://www.facebook.com

approximation is needed. On the other hand, text search is known to be scalable since text IR methods uses efficient data structures such as lexicon, posting lists and boolean operations that make it very efficient for searching in large collections.

In this work, we try to bridge the performance gap between text IR and metric IR by introducing an approximation for top-k using MII (Metric Inverted Index) - an inverted index implementation for metric spaces. We show that by using text IR methods such as lexicon, posting lists and boolean operators we can achieve high scalability for search in metric spaces. Using the MII we can tradeoff between efficiency and effectiveness of our approximated top-k. We show an implementation of a system based on Lucene[2] and report experiments on a real data collection that shows the efficiency of our method compared to other methods.

The paper is organized as follows; we review related work in Section 2. In Section 3, we define metric spaces and state the problem. In Section 4, we describe the MII - Metric inverted index for metric spaces. Then, in Section 5, we show how to use it efficiently for top-k retrieval. Experimental setup and results on a large image collection are given in Section 6 and we conclude with summary and future work in Section 7.

2 Related work

Using Text IR techniques for retrieving multi-media content was already suggested by several works. The early works [9, 5] suggested the use of inverted index with a lexicon containing pre-defined set of around 80000 features which correspond to human perception features. Their reported experiments are done on limited collection of 2500 images, so scalability is not addressed.

[12] facilitates the use of inverted index for indexing N-grams of visual descriptor vectors. Their approach is limited to vectors only and thus can not be generalized to general metric spaces. More recently, [1, 6, 8] showed how to use inverted index for search in images. Specifically, in [1], each image is segmented into several segments and segments are mapped to lexicon entries. These works on images only and not on general metric spaces.

Another line of related works discuss methodologies of indexing and accessing data in metric spaces and are covered in details in [7, 14]. These techniques are general and work for every metric space; however, their main disadvantage is that their complexity increase linearly with the collection size, thus, they are not scalable and not appropriate to large-scale collections. Each of the techniques partition the metric space differently; specifically, [13] considered the space as a ball and applied ball-partitioning. An orthogonal approach is the GHT [10], where two reference objects (pivots) are arbitrarily chosen and assigned to two distinct objects subsets. All objects are assigned to the subset containing the pivot which is nearest to the object itself. In contrast to ball partitioning, the

[2] http://lucene.apache.org

generalized hyperplane does not guarantee a balanced split. Finally, [2, 3] present another way of partitioning the data having balanced split.

3 Definitions and Problem Statement

3.1 Definitions

Metric Spaces A metric space is an ordered pair (S, d), where S is a domain and d is a distance function $d : S \times S \rightarrow \mathbb{R}$ between any two objects in S. Usually, the smaller the distance is, the more similar the objects are. The distance function d must satisfy non-negativity, reflexibility, symmetry and triangle inequality. Assume a metric space (S, d), a collection $C \subseteq S$ of objects, and a query $Q \in S$. The *best-k results* for the query are the k objects with the smallest distance to the query object. For compatibility with IR scoring we transform the distances to *scores* and we assign to each object D a score $sd(Q, D)$ in the range [0,1] such that objects with smaller distance get higher score.

Top-k problem Assume m metric spaces $\{(S_i, d_i))\}_{i=1}^{m}$, a query $Q \in \prod_{i=1}^{m} S_i$, and an aggregate function $f : R^m \rightarrow R$. The problem of top-k is to retrieve the best k objects that are similar to the given query, namely those objects D with the highest aggregate score $f(sd_1(Q, D), sd_2(Q, D), \ldots, sd_m(Q, D))$ over all m features.

Measures

- **effectiveness** To measure effectiveness of a retrieval engine we use the standard Precision and MAP (Mean average Precision) [11] measures. The precision is used to measure the percentage of documents *retrieved* that are *relevant*. In metric spaces each object has some distance to the query (and thus some level of relevancy). Thus, we define $MAP@k$ which is the same as MAP but we take the *relevant* results to be the real top-k objects.
- **efficiency** Let us assume that the cost of a single distance computation between any two objects in any of the metric spaces is c_d. The efficiency of a retrieval engine on metric spaces is measured by the total computation cost $(c_d \times N_d)$, where N_d is the total number of distance computations needed to answer a top-k query.

3.2 Problem Statement

Assume $\{(S_i, d_i))\}_{i=1}^{m}$ metric spaces and a cost c_d for a single distance computation. Our goal is to return the best approximation (measured by effectiveness) to the top-k query while minimizing the total computation cost (efficiency measure).

4 Indexing

We assume m metric spaces denoted by $\{(S_1, d_1) \cdots (S_m, d_m)\}$ and a collection of N objects each having m features $F_1 \cdots F_m$ such that $\forall i \in [1, m], F_i \in (S_i, d_i)$.

We describe now a Metric Inverted Index for metric spaces. Similarily to an inverted text index it contains two main parts: a *lexicon* of features in the collection and *posting lists* containing for each feature the list of documents in which the feature appear.

4.1 Lexicon candidate selection

In text IR, the size of the lexicon is substantially smaller than the size of the collection since terms tend to appear in many documents. This is not always the case for metric spaces. For example, low level image features such as e.g. a Scalable Color can be reprensented by a vector of 256 integers. Since the space of possible values for each feature is huge, the size of a lexicon will be in the order of the collection size. Consequently, it would be impossible to store all features in the lexicon as in text IR.

Thus we would like to select candidates for the lexicon such that the max number of lexicon entries will be at most some parameter l. Note that we keep all the different features from all the metric spaces in the same lexicon. A naive approach would consist by ramdomly selecting l/m documents per feature. The m features extracted from these documents will constitute the lexicon terms. This approach can be widely improved by applying some clustering algorithms on the extracted features using the distance function such that selected candidates spans the metric space. In the present work, we use K-Means clustering algorithm [4].

For each feature, we choose randomly $M = l/m$ centroids. Note that generally $M << N$. K-Means algorithm is run iteratively by mapping each feature to the closest centroid using the distance function. Then new centroids are selected and the process continues until fixed clusters are reached, namely no feature was moved to another centroid in a full iteration. The lexicon elements are then the clusters' centroids. K-Means is run for each of the m features and we get a lexicon of l terms.

4.2 Index creation

Once the lexicon entries are selected we need to create the posting lists. Let us denote the terms of the lexicon by $(F : v)$ where F is the feature name and v is its value. Assume an object D with features $D = \{(F_1 : v_1), (F_2 : v_2), \ldots, (F_m : v_m)\}$.

Most of the features $(F_i : v_i)$ are not part of the lexicon so we need to select from the lexicon terms that best represent our object. This is done by selecting for each feature $(F_i : v_i)$, the n nearest lexicon terms (n is a parameter) according to the distance function. In order to determine these nearest terms efficiently, the lexicon is stored in an M-tree [2]. After this stage, D can be represented as:

$D = \{(F_1 : v_{11}), \ldots (F_1 : v_{1n}), \ldots, (F_m : v_{m1}), \ldots, (F_m : v_{mn})\}$,
where $(F_i : v_{ij})$ are all the lexicon terms.

5 Retrieval stage

Given a query $Q = \{(F_1 : qv_1), (F_2 : qv_2), \ldots, (F_m : qv_m)\}$ we describe now the query processing done to retrieve the best approximation of the top-k while minimizing the computation cost as defined in Section 3. Query processing is done into three steps: term selection, Boolean constraint filtering and scoring.

5.1 Term selection:

Similarly to the indexing process described in Section refindex, we first find for each query feature, the n closest lexicon terms. Consequently, Q is expanded to $Q' = \{(F_1 : qv_{11}), \ldots (F_1 : v_{1n}), \ldots (F_m : qv_{m1}), \ldots, (F_m : qv_{mn})\}$.
Q' contains $m \cdot n$ terms all from the lexicon.

5.2 Boolean Constraint Filtering:

We look now at the posting lists of the $m \cdot n$ terms in Q' and we want to select the best approximation to the top-k out of the objects in those posting lists. We present two strategies to select the best results:

Strict-Query mode - recall that we have m features in the query and for each feature we have n posting lists of the closest lexicon entries. In the strict mode we would like to return results only if they contain all the m features. To achieve that we create a conjunction of m clauses C_i, where clause C_i is a disjunction of the n terms found for query feature (F_i, qv_i). Experiments demonstrate that this occurs very rarely in real configuration. Formally,

$$\bigcap_{i=1}^{m} \cup_{j=1}^{n} (F_i : qv_{ij}) \tag{1}$$

Fuzzy-Query mode - in this mode, we consider all objects that appear in at least 2 features. To achieve that, we create $\binom{m}{2}$ conjunction clauses for each pair of features and then we create a disjunction between those clauses. Formally,

$$\bigcup (\bigcap_{i=1}^{m} (\cup_{j=1}^{n} (F_i : v_{ij}), \cup_{j=1}^{n} (F_k : v_{kj})), i \neq k) \tag{2}$$

5.3 Scoring

At the end of the filtering step we have a list of documents and we would like to rank them by relevance to the query. Given an aggregate function f, we compute for each returned document D, its aggregate score to the query Q given by $f(sd_1(Q, D), sd_2(Q, D), \ldots, sd_m(Q, D))$. Our top-k result is the k documents with the highest aggregate score.

6 Experiments

In this section, we analyse the MII under various settings with focus on efficiency and effectiveness. Specifically, we experimented variations of lexicon size and number of nearest terms. In addition, we compare the MII to the M-Tree data structure, one of the state-of-the-art data structure for metric spaces[3].

6.1 Collection description

Our experiments were done on a collection of 160, 000 images crawled from Flickr. For each image, three features have been extracted - ScalableColor, EdgeHistogram and ColorLayout. We used L_1 (Manhattan distance) for the ScalableColor and EdgeHistogram features and L_2 (Euclidean distance) for the ColorLayout. The aggregate function f is simple sum. We randomly choose 180 images from the collection to serve as the queries; for each we computed the MAP and the number of comparisons. The retrieval was done using the ***Fuzzy-Query mode***.

6.2 Results

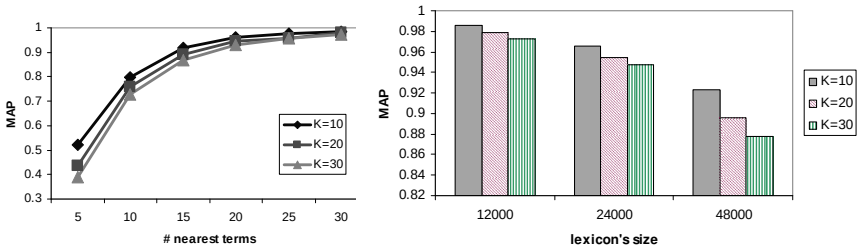
Effectiveness We present the effectiveness of the MII approach using the MAP@k measure. Figure 1(a) shows the MAP@k values (for k=10, 20 and 30) vs. the number of nearest terms for index with lexicon size of $l = 12000$ (4000 entries for each feature). We can see that when the number n of nearest terms increases, the MAP@k increases as well because more candidates pass the filtering step. For example for $n = 10$ the MAP@10 is 0.798 while for $n = 30$ the MAP@10 is 0.98.

Figure 1(b) shows the MAP@k values relatively to the lexicon size for a fixed number of nearest terms $n = 30$. Similarly, we can see that reducing the size of the lexicon increases the MAP@k because for smaller lexicon size more elements are in the posting list of each term and thus more good candidates pass the filtering step. This improvement in the MAP@k results in a more expensive cost computation as we will see in the efficiency subsection and thus we can tradeoff between effectiveness and efficiency.

Efficiency The efficiency is measured by the number of distance computations that is calculated as follow; assume m different features per object. For each feature f_i, we use M-Tree to find the n nearest terms in the lexicon. We designate this cost as $lookup(f_i)$. Then assume N documents are returned from the Boolean constraint filtering so we need $N \times m$ computations to score them. The cost of the retrieval is thus $\sum_{i=1}^{m} lookup(f_i) + N \times m$.

Figure 2 shows the efficiency results for various sizes of lexicon using $n = 30$ nearest terms and for M-Tree. The M-Tree was created for the 160K collection

[3] The M-Tree implementation we used has been taken from http://lsd.fi.muni.cz/trac/mtree



(a) (b)

Fig. 1. MAP value for different k's.(a) Lexicons' size =12000, varied number of nearest terms(b) number of nearest term =30, varied size of lexicon

with a distance function using a pre-cooked combination of the 3 distance functions of the individual features. All our settings achieved much less distance computations than M-tree where our best setting ($l = 12000$) achieved improvement of about 90% efficiency over M-tree.

We can also see the tradeoff between effectiveness and efficiency. For lexicon size $l = 3000$ the computation cost is 3 times more then for $l = 48000$, whereas the MAP@30 was 0.997 and 0.877 respectively. The best cost-effective setting was with lexicon size $l = 12000$, with almost best efficiency and almost best effectiveness (with MAP= 0.985). For lexicons of size $l = 12000$ and $l = 48000$ the cost is almost the same and this can be explained by the fact that the number of lexicon lookups for the larger lexicon is still dominant over the number of the final number of documents that passed the filtering step.

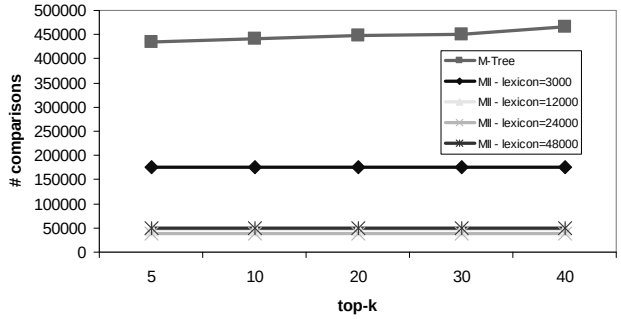


Fig. 2. # distance computations for different MII and M-tree vs. varied k value

In addition, our approach does not depend on k while for M-Tree the number of computations increases when k increases. Moreover with our approach we can combine the m features using varying weights and aggregation per query while with the M-tree approach the aggregate function is fixed at indexing time.

7 Conclusions and Future Work

In this paper we showed a new approximation approach for top-k over metric spaces using MII - Metric Inverted Index. Our experiments show that we achieve very good approximation (MAP=98%) with vast efficiency improvement (90%) over state of the art methods. Furthermore by varying the size of the MII lexicon and the number of nearest terms we can tradeoff efficiency vs effectiveness. For future work we plan to experiment with larger scale collections (in the order of million objects) and to exploit MII to combine search over text and audio-visual content.

8 Acknowledgment

We would like to thank Michal Batko for support in experiments with M-tree and to CNR for supplying the test collection.

References

1. G. Amato, P. Savino, and V. Magionami. Image indexing and retrieval using visual terms and text-like weighting. In *DELOS Conference*, 2007.
2. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, August 1997.
3. V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. D-index: Distance searching index for metric data sets. *Multimedia Tools Appl.*, 21(1), 2003.
4. J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.
5. H. Muller, D. McG. Squire, W. Muller, and T. Pun. Efficient access methods for content-based image retrieval with inverted files.
6. R. Frias R. Jesus, R. Dias and N. Correia. Multimedia information retrieval: "new challenges in audio visual search". *SIGIR Forum*, 41(2):77–82, 2007.
7. Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
8. Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV '03*, 2003.
9. D. Squire, W. Muller, H. Muller, and J. Raki. Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback. In *SCIA*, June 1999.
10. Jeffrey K. Ulmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.
11. C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.
12. P. Wilkins, P. Ferguson, A.F. Smeaton, and C. Gurrin. Text based approaches for content-based image retrieval on large image collections. In *EWIMT*, 2005.
13. P. Yianilos. Excluded middle vantage point forests for nearest neighbor search, 1999.
14. P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.