Information Processing and Management 45 (2009) 401–412

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

Effective top-*k* computation with term-proximity support

Mingjie Zhu [a,*], Shuming Shi [b], Mingjing Li [a], Ji-Rong Wen [b]

[a] University of Science and Technology of China, Hefei, 230026, China
[b] Microsoft Research Asia, Beijing 100080, China

ARTICLE INFO

Article history:
Received 24 March 2008
Received in revised form 2 April 2009
Accepted 3 April 2009
Available online 1 May 2009

Keywords:
Top-*k*
Dynamic index pruning
Term-proximity
Document structure
Proximity-Probe
Non-linear ranking function
Approximate top-*k*

ABSTRACT

Modern web search engines are expected to return the top-*k* results efficiently. Although many dynamic index pruning strategies have been proposed for efficient top-*k* computation, most of them are prone to ignoring some especially important factors in ranking functions, such as term-proximity (the distance relationship between query terms in a document). In our recent work [Zhu, M., Shi, S., Li, M., & Wen, J. (2007). Effective top-*k* computation in retrieving structured documents with term-proximity support. In *Proceedings of 16th CIKM conference* (pp. 771–780)], we demonstrated that, when term-proximity is incorporated into ranking functions, most existing index structures and top-*k* strategies become quite inefficient. To solve this problem, we built the inverted index based on web page structure and proposed the query processing strategies accordingly. The experimental results indicate that the proposed index structures and query processing strategies significantly improve the top-*k* efficiency. In this paper, we study the possibility of adopting additional techniques to further improve top-*k* computation efficiency. We propose a Proximity-Probe Heuristic to make our top-*k* algorithms more efficient. We also test the efficiency of our approaches on various settings (linear or non-linear ranking functions, exact or approximate top-*k* processing, etc.).

1. Introduction

Major commercial web search engines (Google,[1] Yahoo!,[2] Live Search,[3] etc.) have grown to index billions of pages. In spite of such large amount of data, end users expect search results being retrieved quickly with high accuracy. In web search, as the top few results, instead of all relevant documents, are often required, it is possible to speed up the search process by skipping the relevance computation of some documents. Efficient top-*k* computation, which aims to return the correct top-*k* results as quickly as possible, is therefore critical for web search efficiency. Various dynamic index pruning mechanisms (Anh & Moffat, 2006a,b; Fagin, 2002; Long & Suel, 2003) have been proposed for this purpose.

In most papers studying efficient top-*k* computation, ranking functions in the following form are usually assumed

$$F(D,Q) = \alpha \cdot G(D) + \beta \cdot T(D,Q) = \alpha \cdot G(D) + \beta \cdot \sum_{t \in Q} \omega_t \cdot T(D,t). \tag{1.1}$$

Here $F(D,Q)$ represents the overall relevance score of document D to query Q. $F(D,Q)$ consists of two parts: the (query-independent) static rank (e.g. PageRank, Brin & Page, 1998) of the document $G(D)$, and the relevance score of D to query $QT(D,Q)$.

* Corresponding author. Address: 4F, Sigma Center, No. 49 Zhichun Road, Beijing 100080, China. Tel.: +86 10 5896 3064; fax: +86 10 8809 7306.
E-mail address: mjzhu@ustc.edu (M. Zhu).

[1] http://www.google.com.
[2] http://search.yahoo.com.
[3] http://www.live.com.

0306-4573/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ipm.2009.04.002

$T(D,t)$ is the term-score of D and query term t, computed via a term-weighting function (e.g. BM25, Robertson, Walker, Beaulieu, & Gatford, 1994). And α, β and are parameters satisfying $\alpha + \beta = 1$ and $\sum_{t \in Q} = 1$. The above formula is monotonic with respect to document static rank $G(D)$ and term score $T(D,t)$. Fagin (2002) proves that with the monotonic property, an efficient top-k algorithm exists.

In addition to those factors in Formula (1.1), the search quality of modern search engines depends heavily on some other evidence, such as *document structure* and *term-proximity*. *Document structure* means that a web page often comprises multiple fields (title, URL, body text, etc). It has been proved that an appropriate usage of document field structure can improve search results effectively. *Term-proximity* demonstrates how close query terms appear in a document. Intuitively, one document with high term-proximity values (i.e. query terms are near in the document) should be more relevant to the query than another document in which query terms are far away from one another, given that other factors are the same for the two documents. Take query "knowledge management" as an example. It is clear that quite a lot of pages on the web containing both "knowledge" and "management" are actually irrelevant to knowledge management. On the other hand, a document, in which term "knowledge" is always followed by term "management", is much probably related to "knowledge management".

Due to the importance of term-proximity and document structure in modern commercial search engines, efficient top-k computation approaches should be studied by considering these factors. Unfortunately, although quite a few indexing pruning strategies have been proposed, few of them consider term-proximity in the ranking functions. With term-proximity and document structure information being considered, the ranking functions would become more complex (see the analysis in Zhu, Shi, Li, & Wen, 2007). More importantly, term-proximity makes a ranking function NOT monotonic (w.r.t. query term scores) any more. Since term-proximity is determined by the relationship between all query terms rather than one single query term, a document with *low* term score values may have a *high* term-proximity score (and therefore a high overall relevance score), and vice versa. This adds additional challenges to the efficient top-k computation problem.

In our recent work (Zhu et al. 2007) we addressed the problem of efficient top-k computation with term-proximity support. According to our study, most existing top-k strategies become quite inefficient with term-proximity information being included, partially due to the non-monotonicity of the new ranking functions. We then propose two index structures and their corresponding pruning strategies, which efficiently address the top-k problem by utilizing web page structure information.

In this paper, we make a more comprehensive study on this problem in addition to our previous work. First, we study the possibility of adopting additional techniques to further improve top-k computation efficiency. We propose the *Proximity-Probe Heuristic* to further improve the performance. Experimental results show that some ineffective top-k algorithms can get back to work with this heuristic. Second, we study the problem on more settings. In terms of ranking functions, we test and compare the efficiency of different approaches using a non-linear ranking function as well as a linear one. In terms of result quality, we conduct experiments in the case of accurate top-k and approximate top-k, respectively.

The rest of this paper is organized as follows. Section 2 introduces the preliminary knowledge and related efforts. In Section 3, we first introduce how to adopt the Proximity-Probe Heuristic upon the work in Zhu et al. (2007) for more efficient top-k processing; then we exploit the top-k problem with more settings like non-linear ranking function and the approximate top-k results. The experiment results are presented in Section 4. Finally we draw the conclusion in Section 5.

2. Background and related work

In this section, we first give some background concepts related to top-k computation. Then we briefly introduce the traditional top-k approaches for web search and other related work.

2.1. Background

Indexing and ranking are two key components of a web search engine. To support efficient retrieval, a web collection needs to be offline indexed. Given a query, one ranking function is adopted to compute a relevance score for each document based on the information stored in the index. The documents are then sorted by their scores and the top-k documents with the highest scores are returned to end users.

One primary way of indexing large scale web documents is the inverted index, which organizes document collection information into many *inverted lists*, corresponding to different terms. For a term t, its inverted list includes the information (DocIds, occurring positions, etc.) of all documents containing the term. Zobel and Moffat (2006) give a comprehensive introduction to inverted index for text search engines.

2.2. Efficient top-k processing without term-proximity

Various dynamic index pruning techniques have been proposed for efficient top-k computation. They aim to correctly identify the top-k results without completely scanning inverted lists and/or computing the relevance scores of all documents. One common idea shared among most existing efficient query processing approaches is *score upper-bound estimation and early stopping*, which are derived from Fagin's TA algorithm (Fagin, 1996; Fagin, 2002; Fagin, Lotem, & Naor, 2001).

During processing a query, the maximal possible score of all unseen (i.e. un-evaluated) documents is estimated. When the maximal possible score is not greater than the score of the kth document in current top-k list, we can skip processing the remaining documents and return the current top-k results to users. Let S_k be the score of the kth document (in terms of relevance scores) in processing a query, and S_T be the maximum possible score of all un-evaluated documents. Then the early-stopping condition is

$$S_k \geq S_T \tag{2.1}$$

How to organize inverted indexes is the key to index pruning. For example, if documents are naturally ordered by DocIds in the inverted lists, it should be hard to get desired upper-bound estimation as it provides little information about the scoring factors of unseen documents. The primary idea of optimizing the index structures is to put documents with higher scoring factors at the top of an inverted list. So S_T can be lowered down quickly.

For ranking function 1.1, S_T is composed by two parts: the upper bound of static rank and term score.

$$S_T = \alpha \cdot G_{upper} + \beta \cdot \sum_{t \in Q} \omega_t \cdot T_{upper}(t) \tag{2.2}$$

Order by static rank: By sorting inverted index in the descending order of static rank, G_{upper} can be constrained when we access the index as we can expect the subsequent documents have smaller static rank. However, most likely the α value should not be too big as in practical the static rank shall not be the dominant factor for the final relevance score. So the decrease of G_{upper} cannot significantly lower down S_T accordingly. In our previous study, this index structure hardly achieves any performance gain.

Multi-segment Divided by Impact Values: In some work (Anh & Moffat, 2006b; Long & Suel, 2003; Persin, Zobel, & Davis, 1996) a more advanced index structure is proposed. This structure divides an inverted list into multiple segments according to one type of specific term impact values, with documents in each segment ordered by DocIds or static rank. The phrase "term impact" here may refer term scores, term frequency or other factors related to the relevance score of a document to a term. Generally the segment containing high term impact documents is accessed first. So we can have a lower estimation of $T_{upper}(t)$ in the lower segment. While within each segment the descending order of static rank helps decrease G_{upper}. This strategy is popular in web search as it's effective with not too many complex additional efforts.

2.3. Related work

In database community, Fagin (1996, 2002) and Fagin et al. (2001) have a series of fundamental work on efficient top-k computation. As Fagin (2002) states, if multiple inverted lists are sorted by the attributes value, and the combination function is monotonic, an efficient top-k algorithm exists.

In information retrieval area, the work of index pruning could go back to 1980s. Some earlier works are described in (Buckley & Lewit, 1985; Burrows, 1999; Harman & Candela, 1990; Moffat & Zobel, 1996; Persin et al., 1996; Turtle & Flood, 1995). Their main idea is to sort the entries by their contribution to the score of the document and put the important entries in the front of inverted index for early termination. Anh and Moffat (2006a), Kaushik, Krishnamurthy, Naughton, and Ramakrishnan, (2004) have attempted to optimize the index structure in various ways. Persin et al. (1996) propose to partition the inverted list into several parts. When PageRank shows its power in web search, pruning techniques considering PageRank are studied in Long and Suel (2003). Ntoulas and Cho (2007) study the keyword locality and document locality to get an optimized ratio between small index and full index in the parallel processing scenario.

Static index pruning (Büttcher & Clarke, 2006; Carmel et al., 2006; Moura et al., 2005; Persin et al., 1996) aims at reducing index size by keeping only relatively important information of the inverted index. Particularly, Blanco (2008) gives a comprehensive study on the index optimization of information retrieval systems. However, in this paper, we focus on the *dynamic* index pruning problem which skips the computation of some document scores at query execution time.

We noticed a very interesting study of efficient text proximity search (Schenkel, Broschart, Hwang, Theobald, & Weikum, 2007). In this paper, pre-computed and materialized index structures are proposed to boost the top-k processing performance. However, their work is more like a database style implementation and not applicable in real commercial web search scenarios.

Broder, Carmel, Herscovici, Soffer and Zien (2003) treat query evaluation as a two-stage process. In the first stage they retrieve and score the documents containing every query term. If the number of scored documents is greater than k, the process stops; otherwise they continue to look for documents contains only part of the query terms. The principle of giving documents containing all query terms the high priority inspires us to study the similar properties in our proposed Proximity-Probe Heuristic.

Approximate (Fagin, 2002) and probabilistic (Theobald, Weikum, & Schenkel, 2004) top-k computation are also important to web search. By relaxing result quality requirements, query processing efficiency has more space to be improved. Many work in Database area like Theobald et al. (2004) analyze the probabilistic top-k computation based on the property distribution over inverted lists. However, they are based on the conventional scoring functions without considering proximity and web page structure. With more factors involved, it becomes much more complex to make such estimations. Then we give the experimental study of approximate top-k processing approaches under the new settings in Section 4.

3. Term-proximity-enabled top-*k* computation

In this section, we first review our previous work in Zhu et al. (2007) which discusses how to compute top-*k* results efficiently for a typical ranking function where the score of a document is the linear combination of static rank, term-weights, and term-proximity scores. Then we introduce our Proximity-Probe rules and demonstrate how they work together with our existing strategies to further improve search performance. Finally we exploit more factors for top-*k* processing, including the situations with non-linear ranking functions and approximate top-*k* processing.

3.1. Efficient top-k processing by exploiting document structure

In our previous work (Zhu et al. 2007), we mainly consider the following typical ranking function which supports term-proximity and document structure

$$F(D,Q) = \alpha \cdot G(D) + \beta \cdot T(D,Q) + \gamma \cdot X(D,Q) \tag{3.1}$$

where $G(D)$ is the static rank of D, $T(D,Q)$ is the overall term score (by aggregating the scores of all query terms) of D to query Q, and $X(D,Q)$ is the term-proximity score between D and Q. Term score $T(D, Q)$ and term-proximity score $X(D,Q)$ are expressed as follows,

$$T(D,Q) = \sum_{t \in Q} \left(\omega_t \cdot \sum_{F_i \in D} \lambda_i \cdot T(D,F_i,t) \right) \tag{3.2}$$

$$X(D,Q) = \sum_{F_i \in D} \mu_i \cdot X(D,F_i,Q) \tag{3.3}$$

where $T(D, F_i, t)$ is the term t's term-weighting score in field F_i of the document, and $X(D, F_i, Q)$ is the term-proximity score of field F_i relative to Q. One example of term weighting functions is the BM25 formula (Robertson et al., 1994). One example of term-proximity weighting function is in Rasolofo and Savoy (2003). The weights and coefficients (α, β, γ, λ_i, and μ_i) satisfy the following constraints,

$$\alpha + \beta + \gamma = 1; \quad \sum_{t \in Q} \omega_t = 1; \quad \sum_{F_i \in D} \lambda_i = 1; \quad \sum_{F_i \in D} \mu_i = 1$$

In order to adopt the early-stopping rule (Formula (2.1)) to generate the top-*k* results efficiently, the maximal possible score S_T of all unseen documents has to be properly estimated. For the ranking function in Formula (3.1), S_T can typically be estimated as,

$$S_T^- = \alpha \cdot G_{upper} + \beta \cdot \sum_{t \in Q} \omega_t \cdot T_{upper}(t) + \gamma \cdot X_{\max} \tag{3.4}$$

where $X_{\max}$ is the estimated maximum possible term-proximity score.

As term-proximity scores are determined by the distance between query terms, it is not anymore a monotonic function of term weighting scores. Therefore small term weighting scores do not imply a low term-proximity score. It is hard for traditional approaches to find an accurate estimation of the maximum possible term-proximity score $X_{\max}$, because the index structures they adopted are mainly for lowering down the estimation of G_{upper} (i.e., the score upper-bound of static-rank) or $T_{upper}(t)$ (i.e. the maximal possible term score of term t). Therefore in most case, we have no other choices but assuming $X_{\max} = 1.0$.

Our experiments in Zhu et al. (2007) showed that traditional top-*k* algorithms become quite ineffective for the ranking function in Formula (3.1) when the γ value is larger than a threshold. In that paper, we proposed an effective approach by organizing each inverted list into two segments: a fancy segment and a body segment. The body segment contains all the term hits which appear in the body field of documents. While the fancy segment contains the hits of other document fields (title, URL, and anchor text). The documents in each segment are sorted by static rank. We call such an index format as *structured*. This kind of index structure is mentioned in Brin and Page (1998). However, they did not illustrate how to compute *accurate* top-*k* results efficiently using such an index structure. In addition, they sort documents by DocId rather than static-rank. We are motivated to investigate such an index format by the observation that the body text of a web page is typically longer than other fields (e.g., title, URL, and anchor-text) but relatively less important in ranking functions. With such an index structure, a three-phase strategy is adopted to process queries. In the first phase, all documents in the first segment (i.e., the fancy segment) is loaded and evaluated. Because the body scores of documents have not been available in this phase, a candidate documents accumulator A is maintained to record the minimal and maximal score of the documents which have the possibility of being in top-*k*. Then in the second phase, documents in the body segment are sequentially evaluated and the early-stopping condition (Formula (2.1)) is periodically checked. Once the early-stopping condition is satisfied, we know it is safe to skip evaluating all the documents which are not in the accumulator A; and then we enter into the third phase. In the third (and last) phase, the body segment is continually scanned and the documents which are in the accumulator A but not in top-*k* are removed from A, until there are only exactly k documents in A. Please refer to Fig. 1 for the pseudo-codes of query processing. One important operation is to estimate the minimal and maximal score of a document

```
Algorithm: Top-k computation for the fancy-body index structure
    Input: Inverted lists L_1, ..., L_m, for the query terms
    Output: The list of top-k documents

    Variables:
        R: Current top-k list;
        A: The accumulator for candidate docs.
        S_K: The minimal score (i.e. the Kth score) in R

    Clear R and A; Set S_k = 0
    ProcessFancySegs(R, A); //process fancy segments
    ProcessBodySegs(R, A); //process body segments
    return R;
End of algorithm

    //function ProcessBodySegs
    ProcessBodySegs(R, A)
        For each doc d in current segment
            Compute d.minscore and estimate d.maxscore
            if(|R| < k OR d.minscore > S_k)
                R.insert(d);
                if(|R| > k)
                    RemoveSmallest(R);  //Remove the doc with the smallest score
                if(|R|>=k)
                    S_K = the smallest score in R
             EndIf

            if(!A.contains(d) AND d.maxscore > S_k)
                A.insert(d);
            if(A.contains(d) AND d.maxscore <= S_k)
                A.remove(d);

            Compute S_T (by Formula 3.4)
             if(|R| >= k AND S_k >= S_T AND A-R is empty)
                return;
        EndFor
    EndFunc

    //function ProcessFancySegs
    ProcessFancySegs(R, A)
    //The implementation is similar to ProcessBodySegs. Omitted to remove
redundancy
```

Fig. 1. Pruning strategy for the structured index.

d. When we are in the fancy segments, *d*.minscore can be estimated by assuming a zero body score (i.e., no query terms appearing in *d*'s body text). Similarly, we estimate *d*.maxscore by assuming the maximal possible body score a document could have. When come to the body segments, we have enough information to compute the accurate score of a document: *d*.minscore = *d*.maxscore = *d*.score. Once we get the estimated minimal and maximal scores of a document, we check whether it can be added *R*, the current top-*k* list. The document is added into *R* if there are less than *k* elements in the list, or if the minimal score of *d* is larger than the smallest score in the list. S_K may be updated (to become larger) after a new document is added into *R*.

In our settings, the fancy segment contains three fields: title, URL, and anchor text. The title of a web page is the small piece of text included in the <TITLE> HTML tag. The anchor text for a page is the collection of the clickable text when the page is mentioned in other pages. Since the fancy segments are relatively small in size (compared with their corresponding body segments), we could process them very quickly when processing a query. When we come to the body segments, only the term-proximity upper bound of the body segment needs to be estimated. Therefore one main advantage of this index structure is that we are able to get a more precise estimation of $X_{\max}$ when scanning the body segment, as follows,

$$X_{\max} = \mu_{body} \max_{D} X(D, F_{body}, Q) \tag{3.5}$$

As μ_{body} is much less than 1.0 (recall that $\sum_{F_i \in D} \mu_i = 1$), the value of $X_{\max}$ is much less than 1.0 accordingly.

In addition to dividing the inverted lists by document structure, we can further split the body segment into two segments: *body_high*, and *body_low*. The *body_high* segment contains documents with relatively high term weighting scores (i.e. $T(F_{body}, t)$ is larger than a threshold). And the documents with low term weighting scores are in the segment *body_low*. The documents in each segment are still sorted in descending order by static rank. The hybrid index structure is expected to combine the merits of two index structures: *structured*, and *multi-impact*.

Experimental results showed that the query processing strategies based on the above two index structures are able to achieve up to one order of magnitude of performance improvement for typically settings. Please refer to Zhu et al. (2007) for more details of the proposed index structures and their corresponding query processing strategies.

3.2. The Proximity-Probe Heuristic

The approaches in the previous subsection aim to achieve early stopping by adopting document structure information. With early stopping, we are able to reduce the number of documents being accessed or processed. When term-proximity is included in ranking functions, we observed that it is much more time-consuming to compute the term-proximity score of a document than its term-score, because term-proximity score computation needs to access all hits of the query terms in the document and to calculate the distance between hits. If we can avoid the term-proximity score calculation of some documents whose term-scores have been computed, we can save more query processing time. We call a document is *fully* evaluated if both the term score and the term-proximity score of the document is computed. In contrast, if the term score of a document is computed while its term-proximity score computation is skipped, we call this document is *partially* evaluated. Here we study in this subsection a top-k enhancement technique which improves top-k processing efficiency by reducing the number of documents being *fully* evaluated.

Thus we have the following heuristic for accelerating top-k computation,

Proximity-Probe Heuristic: For a specified document D, assume $G(D)$ is the static-rank of the document, $T(D)$ is the aggregated term score (over all the query terms and fields), and $X_{\max}(D)$ is the estimated maximal term-proximity score of the document. Assume $G(D)$ and $T(D)$ is available, it is safe to skip the term-proximity score computation of document D if the following formula is satisfied,

$$\alpha \cdot G(D) + \beta \cdot T(D) + \gamma \cdot X_{\max}(D) \leq S_k \tag{3.6}$$

The correctness of the above heuristic is straightforward. The satisfaction of Formula (3.6) means the maximal possible score of document D is not larger than the score of the kth document in the top-k list. It is therefore not necessary to bother calculating the term-proximity score of the document. Please pay attention that the above estimations are document-dependent.

According to the above heuristic, we can speed up top-k computation by the following way. For a document D in the document score computation process, we first fetch the static rank $G(D)$ and compute the term score $T(D)$; then before actually performing the term-proximity score calculation, we first combine the estimated maximal term-proximity score of the document with the actually computed term scores to get an estimated maximal overall score. If the estimated overall score is less than the minimal score of the kth document, we skip the document without actually computing its term-proximity score.

Now the problem is how to precisely compute $X_{\max}(D)$, the maximal possible term-proximity score of document D. Assuming the *Structured* index format (referring to Section 3.1) is adopted, we have,

$$X_{\max}(D,Q) = \begin{cases} \sum_{F_i \in \{\text{Fields of } D\}} \mu_{F_i} \cdot X_{\max}(D,F_i,Q) & \text{If in the fancy segment} \\ \mu_{body} \cdot X_{\max}(D,F_{body},Q) + \sum_{F_i \in \{\text{Fancy Fields}\}} \mu_{F_i} \cdot X(D,F_i,Q) & \text{If in the body segment} \end{cases} \tag{3.7}$$

where μ_{F_i} is the weight of field i's proximity score, and $X_{\max}(D, Fi, Q)$ is the estimated maximal possible term-proximity score of document D in field i. Refer to Formula (3.3), the proximity score is made up of the individual term-proximity scores in each fields. So the maximal term-proximity score is determined by which fields the index segment contains. For the *Structured* index format, we first access the fancy segment and use the combination of the maximal possible term-proximity scores of all fields as the estimated maximal term-proximity score of the document; then after we processed the fancy segment and come to the body segment, $X_{\max}(D,Q)$ can be estimated as the combination of the maximal possible term-proximity score for the body field with the accurate term-proximity scores of other fields. Then it is similar to the analysis in previous subsection, the *Structured* index also helps when we adopt the Proximity-Probe heuristic to estimate the maximal proximity score.

In order to estimate $X_{\max}(D,Q)$ using Formula (3.7), we need to have an estimation of $X_{\max}(D, F_i, Q)$, the maximal possible term-proximity score of field i in document D. Here we propose two term-proximity score estimation (TPE) rules:

TPE Rule-1: $X_{\max}(D, F_i, Q) = 1.0$

By rule-1, we set the maximal possible score as 1.0 (the maximal possible term-proximity score for all documents) for each field of the document. This is a loose and coarse-grained estimation which does not consider the information about the document we have in hand at the moment.

TPE Rule-2: $X_{\max}(D, F_i, Q) = r(|Q|, m) \cdot 1.0$

In the above rule, $|Q|$ is the query length (i.e., number of query terms), m is the number of distinct query terms occurred in field F_i of document D, and $r(n, m)$ is a function which takes value 1.0 if $m = n$ and value smaller than 1.0 if $m < n$. By studying the characteristics of term-proximity, we find that the maximal possible term-proximity score of a document D can be estimated to be less than 1.0 if we have some knowledge about the document. Intuitively, if we have already known that not all the query terms appear in one specific field of one document, the term-proximity score of the field should be smaller than the maximal possible term-proximity score.

Here we take one simple term-proximity function to illustrate the TPE rule-2.

$$X_{naive}(D, F_i, Q) = \frac{1}{\binom{|Q|}{2}} - \sum_{i<j} \frac{1}{\min_{p_i, p_j}(1 + |(p_j - j) - (p_i - i)|)^2} \tag{3.8}$$

where p_i denotes the offset of ith query term occurring in field F_i of document D. The term-proximity score is calculated by aggregating the inverse square of the distance between query terms for every term pair. Given a query Q containing three query terms $Q = \{t1, t2, t3\}$, assume only two of the three query terms appear in the field F_i of document D_1. According to Formula (3.8), we have $X_{naive}(D_1, F_i, Q) \leqslant 1/3$. Generally, for the term-proximity function in Formula (3.8), it is not hard to prove that,

$$r(n, m) = \binom{m}{2} \Big/ \binom{n}{2}$$

For different term-proximity functions, function $r(n,m)$ may be different. However, for any reasonable term-proximity function, a document should not achieve the maximal term-proximity score if it does not contain all the query terms. Therefore $r(n,m)$ should be less than 1.0 if $m < n$.

3.3. More factors for performance comparison

There are different settings for the top-k problem we discussed. Here we exploit two popular variants. First we study the problem when the ranking function is non-linear; then we raise the problem of approximate top-k processing.

3.3.1. Non-linear ranking functions

Besides the popular linear ranking function like Formula (3.2), there could be also non-linear ranking functions. Here we propose a simple heuristic ranking function as Formula (3.5) to study the term-proximity problem,

$$F(D, Q) = (G(D) + \alpha_G) * (T(D, Q) + \gamma \cdot X(D, Q)) \tag{3.9}$$

It is not hard to verify that all of the top-k approaches already discussed can be adapted to support the above non-linear ranking function. With such a ranking function, the potential max score S_T for unknown documents is like Formula (3.10). Similar to the situations in the linear ranking function, the term-proximity score factor $\gamma \cdot x_{\max}$ prevents us from estimating S_T^- effectively in traditional top-k strategies.

$$S_T^- = (\alpha_G + g_{\max}) \cdot (t_{\max} + \gamma \cdot x_{\max}) \tag{3.10}$$

However, with the *structured* index, the expression of S_T is changed from Formula (3.10) to

$$S_T^- = (\alpha_G + g_{\max}) \cdot (\lambda_{body} \cdot t_{\max} + \gamma \cdot \mu_{body} \cdot x_{\max}) \tag{3.11}$$

Then it is easier to lower down the estimated value of S_T^-. We give the experimental study with such a non-linear ranking function in Section 4.

3.3.2. Approximate top-k vs. accurate top-k

It is good for a system to return "accurate" top-k results efficiently. There are, however, many circumstances (especially in large scale web search) where approximate top-k (Burrows, 1999; Fagin et al., 2001) results are enough to meet the users' information needs. For two items scored 0.914 and 0.915 respectively, users may not care about the relative order between them. Strictly speaking, item scores computed by most (if not all) ranking functions are actually just an *approximate* measure of the "*real*" score. Any top-k results ordered by such scores are therefore not strictly accurate.

According to Fagin et al. (2001), k items $r_1, r_2, \ldots, r_k$ is an θ-*approximation* to the top-k results if and only if

$$\begin{aligned} &\theta \cdot t(r_i) \geq t(r_{i+1}) \quad \forall 1 \leq i < k \\ &\theta \cdot t(r_k) \geq t(r) \quad \forall\ r \text{ not in } \{r_1, r_2, \ldots, r_k\} \end{aligned} \tag{3.12}$$

where $t(r)$ denotes the score of item r.

Here parameter θ indicates how fuzzy the ranking results can be tolerated. When $\theta = 1$, we have the exact or accurate top-k results. We will compare, via experiments, the performance of different approaches by varying θ values.

4. Experiments

We now describe the experimental setup and analyze the results. In Subsection 4.1, we discuss the experimental study of the top-k processing with our proposed index structure for different settings. In Subsection 4.2, we present and evaluate the performance of our proposed structure with the Proximity-Probe Heuristic.

Datasets: We use two TREC[4] test collections in the experiment. The GOV collection consists of about 1.25 million web pages crawled from web sites in the ".gov" domain in 2002; and the GOV2 corpus is a crawl of .gov sites in early 2004 with around 25

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

Table 1
Query sets.

	2004mixed	2004tera
# of queries	225	50
Query types	np, hp, td	np, hp, td
Collection	GOV	GOV2
Avg query length	3.43	3.16

td: topic distillation; **np**: named page finding; **hp**: home page finding.

million web pages (Refer to http://www.ir.dcs.gla.ac.uk/test_collections/ for more information about the two collections). For corpus GOV, we only keep html pages (about 1 million), while documents of other types (such as pdf, doc, etc) are discarded. For the GOV corpus, we conduct our experiments on query sets 2004mixed. For the GOV2 corpus, we perform experiments on query set 2004tera which was used in the Terabyte track of TREC in 2004 and 2005. Details of the query sets are shown in Table 1. We choose data collections of different sizes and queries of different types. This allows us to test and compare the performance of various index structures and pruning strategies by various settings.

Hardware and Software Environment: For the GOV dataset, experiments are carried out on a single machine with Dual 3.06 GHz Intel Xeon CPU, 4G RAM and 600 GB local SATA disk. While for corpus GOV2, we distributed its documents to 5 machines (via URL hashing), with each machine having about 5 million documents indexed. The 5 nodes exchange link information by network communication, based on which each document's static rank is computed and its anchor text is accumulated. Only one CPU per machine is used in the pruning experiments, i.e. we are running a single-thread program on each machine.

Index Structures: We compare 4 kinds of index structures as follows.

BASE: The documents in each inverted list are sorted by DocId and no pruning strategies are performed.
PR-1: The documents in each inverted list are sorted by PageRank. In Subsection 2.2, it is the *Order by Static Rank*.
IMPACT-2: Each inverted list has two segments: a high score segment, and a low score segment. Documents in each segment are sorted by PageRank. In Subsection 2.2, it is the *Multi-segment Divided by Impact Values*.
STRUCTURED: Each inverted list contains a fancy segment and a body segment, with documents in both segments sorted by PageRank.

Algorithm and Parameters: Two ranking functions are adopted in the experiments: the linear ranking function expressed in Formula (3.1), and non-linear function in Formula (3.9). In both ranking functions, the term score for each field is computed via the BM25 formula (Robertson et al., 1994) with parameters $k_1 = 1.0$ and $b = 0.66$. The field scores are aggregated like in Song et al. (2004). We vary some key ranking function parameters (listed in Table 2) in experiments to compare the performance of different approaches.

Evaluation Metrics: There are basically two kinds of metrics for evaluating a top-k algorithm: search quality (P@10, average precision, etc.), and processing performance (average query processing time, throughput, etc.). As the top-k processing algorithms are required to return the exact or θ-variant results as the baseline methods. So the search result quality is not our concern and we focus on the measure of the query processing performance. However, we list a group of search quality results in Table 3 for reference purpose. Please be noticed that the parameters are not specifically optimized for the highest quality and we did not include some useful features like HostRank and advanced URL scoring in Song et al. (2004).

Table 2
Variables and their values tested in the experiments.

Parameters	Description	Values
k	Number of results required	1, 5, 10
α	The weight of static rank weight (in Formula (3.1))	0, 0.1, 0.2, 0.5
γ/β	The weight of term-proximity scores relative to that of term weighting scores (in Formula (3.1))	0, 0.1, 0.2, 0.4, 0.5, 1
w_{body}	Body weights relative to the weight of fancy fields	0.5, 1, 2, 4

Table 3
Search result quality (Dataset: GOV; Query set: 2004mixed; $\alpha = 0.2$; $w_{body} = 1.0$; $\gamma/\beta = 0.5$.)

Ranking function	MAP	MRR	P@10
Linear	0.333	0.458	0.116
Non-linear ($\alpha_G = 2.0$)	0.323	0.459	0.115

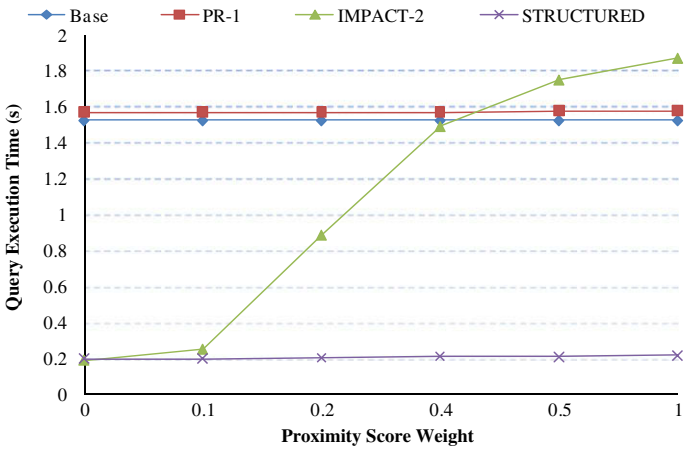

Fig. 2. Performance comparison of various top-k algorithms, for different term-proximity weights (Dataset: GOV2; Query set: 2004tera; $k = 10$; $\alpha = 0.2$; $w_{body} = 1.0$).

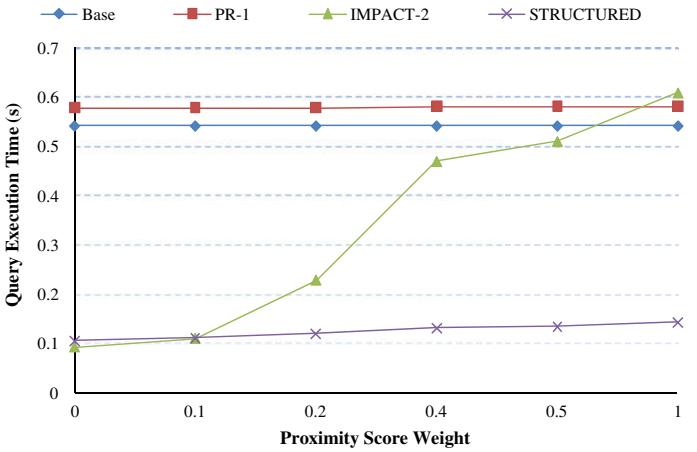

Fig. 3. Performance comparison of various top-k algorithms, for different term-proximity weights (Dataset: GOV; Query set: 2004mixed; $k = 10$; $\alpha = 0.2$; $w_{body} = 1.0$).

4.1. With and without term-proximity

We evaluate the performance of various index structures and pruning strategies before and after term-proximity information is considered in ranking functions. Figs. 2 and 3 show the average query processing time of various algorithms for some selected term-proximity weight values. The results are acquired by using query set 2004tera (corresponding to dataset GOV2), and 2004mixed (corresponding to the GOV dataset) respectively. Some observations can be made from these results. First, although the IMPACT-2 approach behaves well without term-proximity, its performance drops rapidly when the weights of term-proximity get larger. When the term-proximity weight is large enough, IMPACT-2 even performs worse than the baseline. Second, the performance of STRUCTURED is significantly better than the others when term-proximity weight is bigger than a threshold (0.1 in the figures); it could save more than 80% query execution time for all term-proximity weights. Third, no significant performance gains can be acquired with the PR-1 strategy. This is consistent with the statements in Long and Suel (2003). PR-1 can improve performance only when the weight of static rank is extremely high.All experimental results demonstrated in the above are for accurate top-k. To evaluate the efficiency of our proposed strategies for approximate top-k processing, we set different values for the approximate factor θ (referring to Section 3.3.2). Generally, high θ values make the stop condition easier to be satisfied, which will improve top-k efficiency. However, inaccurate results may be returned with large θ values. Fig. 4 shows the performance of different top-k approaches using the linear ranking function, with θ values from 1.0 to 1.2. As predicted, all pruning strategies benefit from larger θ values. We are confirmed with the fact that our newly-proposed approaches are constantly superior to existing top-k strategies for all θ values.

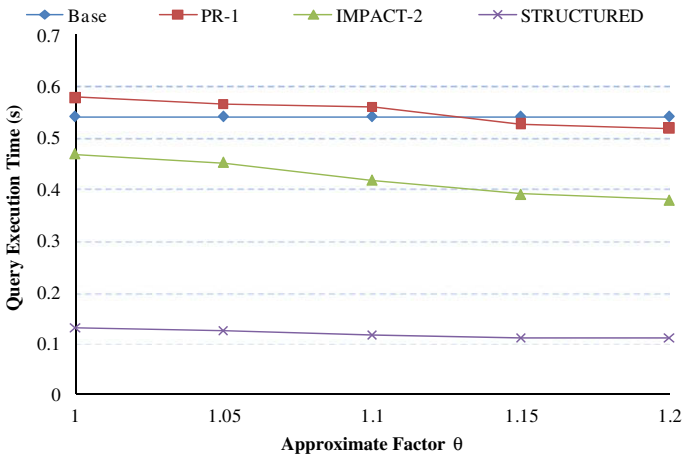

Fig. 4. Performance of approximate top-k processing (Dataset: GOV; Query set: 2004mixed; Ranking function: Linear; $k = 10$; $w_{body} = 1.0$; $\alpha = 0.2$; $\frac{\gamma}{\beta} = 0.5$).

4.2. Evaluate the Proximity-Probe Heuristic

In Subsection 3.2, we propose the Proximity-Probe Heuristic specifically aiming to reduce the number of documents whose term-proximity scores are computed. Here we conduct experiments to verify whether the heuristic can give real performance gains.

In Table 4, we list the average query execution time with different index structures with or without the Proximity-Probe Heuristic. In the first result column, the results without adopting the Proximity-Probe Heuristic are listed. The other two columns list the results after the heuristic being adopted. We can see from the table that the performance improvement is remarkable after the heuristic is utilized. It can also be observed that the Proximity-Probe Heuristic boosts more for the index structures with poor performance before. Specifically, the average query execution time of PR-1 drops from around 0.508 s to less than 0.1 s. This makes a big difference as we can achieve significant improvement on the simple PR-1 structure. For IMPACT-2, we can also achieve over 50% of advance in the performance. For previously well performed STRUCTURED we can still save 20% of average query execution time with the Proximity-Probe Heuristic. We can see that TPE Rule-2 gives further improvement over TPE Rule-1. Please note that it's hard to apply TPE rule-2 to the IMPACT-2 index structure. As for the IMPACT-2, one document appear in one term's high term impact segment may appear in another term's low segment. When processing one segment, we may have one document's partial hit information and it is hard to know the m value in TPE Rule-2.

As the Proximity-Probe Heuristic is adopted to save the processing time by skipping unnecessary term-proximity computations, we list the average proximity calculation times in Table 5. It is very clear that most term-proximity computations are skipped with the Proximity-Probe Heuristic. And TPE Rule-2 can save more proximity calculations than TPE Rule-1. The most representative case is in the second row, without the Proximity-Probe Heuristic, PR-1 has to compute the term-proximity score for almost every document like BASE. However, we find that actually more than 99% of the calculations can be avoided after the Proximity-Probe Heuristic is applied. That's why PR-1 has the most significant performance improvement (even performs better than IMPACT-2). Another interesting observation is that, with TPE Rule-2, PR-1 does even less proximity calculations than STRUCTURED, while taking longer query execution time. This is because PR-1 still has to access much more documents for term score evaluation, although the time for proximity calculation is reduced.

In Fig. 5, we give the experimental results on the non-linear ranking function discussed in Section 3.3.1 with $\alpha_G = 2.0$. Fig. 5 shows the average query processing time of various algorithms for some selected term-proximity weight values. Unlike the linear ranking function, we can see that the performance of IMPACT-2 and STRUCTURED are comparable when the proximity score weight is smaller than 0.5. Besides, we find the Proximity-Probe Heuristic gives extra performance gain for

Table 4
Average query execution time (s) with/without Proximity-Probe Heuristic (Dataset: GOV; Query set: 2004mixed; $k = 10$; $\alpha = 0.2$; $w_{body} = 1.0$).

Index structures	Without Proximity-Probe Heuristic	With Proximity-Probe Heuristic	
		TPE Rule-1	TPE Rule-2
BASE	0.538	0.538	0.538
PR-1	0.508	0.089	0.075
IMPACT-2	0.386	0.224	N/A
STRUCTURED	0.048	0.037	0.033

Table 5
Average term-proximity calculation times with/without Proximity-Probe Heuristic (Dataset: GOV; Query set: 2004mixed; $k = 10$; $\alpha = 0.2$; $w_{body} = 1.0$).

Index structures	Without Proximity-Probe Heuristic	With Proximity-Probe Heuristic	
		TPE Rule-1	TPE Rule-2
BASE	178,617	178,617	178,617
PR-1	168,173	4890	271
IMPACT-2	63,624	3494	N/A
STRUCTURED	27,807	2858	1755

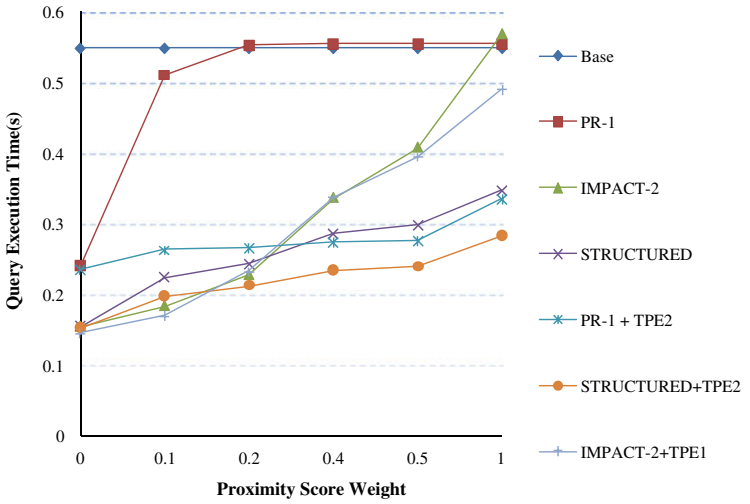

Fig. 5. Performance comparison of various top-k algorithms, for different term-proximity weights (Dataset: GOV; Query set: 2004mixed; Ranking function: non-linear; $\alpha_C = 2.0$; $k = 10$; $w_{body} = 1.0$).

these approaches, especially for PR-1. The results in Fig. 5 indicate that for this naïve non-linear ranking function, our strategies are still effective.

The above results prove that we can save more computational efforts by adopting the Proximity-Probe Heuristic. And the significance is that we can make the traditional index structure which has no pruning effect get back to work with this heuristic.

5. Conclusions

In summary, when ranking functions are fortified with term-proximity factors, top-k problem becomes much more difficult due to the fact that the ranking functions are not monotone any more. In our previous study, most traditional index structures (and their corresponding pruning strategies) had bad performance. And it is interesting that simply splitting index according to document structure can achieve good performance in most settings. Our additional study on non-linear ranking function and approximate top-k processing shows similar conclusions. In this paper, we make a further step to reduce unnecessary proximity computations with the Proximity-Probe Heuristic. With this heuristic, we not only improve the performance of our strategies, but also make the traditional static rank ordered inverted index get back to work.

References

Anh, V. N., & Moffat, A. (2006a). Pruned query evaluation using pre-computed impacts. In *Proceedings of 29th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 372–379).

Anh, V. N., & Moffat, A. (2006b). Pruning strategies for mixed-mode querying. In *Proceedings of 2006 CIKM international conference on information and knowledge management* (pp. 190–197).

Blanco, R. (2008). *Index compression for information retrieval systems*. PhD thesis.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual (Web) search engine. *Computer Networks and ISDN Systems, 30*(1–7), 107–117.

Broder, A., Carmel, D., Herscovici, M., Soffer, A., & Zien, J. (2003). Efficient query evaluation using a two-level retrieval process. In *Proceedings of the twelfth international conference on Information and knowledge management* (pp. 426–434).

Buckley, C., & Lewit, A. (1985). Optimization of inverted vector searches. In *Proceedings of 8th annual SIGIR conference* (pp. 97–110).

Burrows, M. (1999). *Method for parsing, indexing and searching world-wide-web pages*. US Patent 5,864,863.

Büttcher, S., & Clarke, C. (2006). A document-centric approach to static index pruning in text retrieval systems. In *Proceedings of the 15th ACM international conference on information and knowledge management CIKM '06* (pp. 182–189).

Carmel, D., Cohen, D., Fagin, R., Farchi, E., Herscovici, M., Maarek, Y., & Soffer, A. (2001). Static index pruning for information retrieval systems. In *Proceedings of the 24th ACM SIGIR conference on research and development in information retrieval* (pp. 43–50).
Fagin, R., Lotem, A., & Naor, M. (2001). Optimal aggregation algorithms for middleware. In *Proceedings of the twentieth ACM symposium on principles of database systems* (pp. 102–113).
Fagin, R. (1996). Combining fuzzy information from multiple systems. In *Proceedings of the fifteenth ACM symposium on principles of database systems* (pp. 83–99).
Fagin, R. (2002). Combining fuzzy information: An overview. *SIGMOD Record, 31*(2), 109–118.
Harman, D., & Candela, G. (1990). Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *Journal of the American Society for Information Science, 41*(8), 581–589.
Kaushik, R., Krishnamurthy, R., Naughton, J., & Ramakrishnan, R. (2004). On the integration of structure indexes and inverted lists. In *Proceedings of the 2004 ACM SIGMOD* (pp. 779–790).
Long, X., & Suel, T. (2003). Optimized query execution in large search engines with global page ordering. In *Proceedings of the 29th international conference on very large data bases* (pp. 129–140).
Moffat, A., & Zobel, J. (1996). Self-indexing inverted files for fast text retrieval. *TOIS, 14*(4), 349–379.
Moura, E., Santos, C. F., Fernandes, D. R., Silva, A. S., Pável Calado, & Nascimento, M. A. (2005). Improving web search efficiency via a locality based static pruning method. In *Proceedings of the 14th international conference on world wide web* (pp. 235–244).
Ntoulas, A., & Cho, J. (2007). Pruning policies for two-tiered inverted index with correctness guarantee. In *Proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 191–198).
Persin, M., Zobel, J., & Davis, R. (1996). Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science, 47*(10), 749–764.
Rasolofo, Y., & Savoy, J. (2003). Term proximity scoring for keyword-based retrieval systems. In *Proceedings of the 25th European conference on IR research* (pp. 207–218).
Robertson, S. E., Walker, S., Beaulieu, M., & Gatford, M. (1994). Okapi at TREC-3. In *Proceedings of TREC-3, November 1994*.
Schenkel, R., Broschart, A., Hwang, S., Theobald, M., & Weikum, G. (2007). Efficient text proximity search. In *SPIRE 2007, LNCS* (pp. 287–299, vol. 4726).
Song, R., Wen, J., Shi, S., Xin, G., Liu, T., et al. Microsoft research Asia at web track and terabyte track of TREC 2004. In *2004 Text REtrieval Conference (TREC'04)*.
Theobald, M., Weikum, G., & Schenkel, R. (2004). Top-*k* query evaluation with probabilistic guarantees. In *Proceedings of the 30th VLDB conference* (pp. 648–656).
Turtle, H., & Flood, J. (1995). Query evaluation: Strategies and optimizations. *Information Processing and Management, 31*(6), 831–850.
Zobel, J., & Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys, 38*(2), 20.
Zhu, M., Shi, S., Li, M., & Wen, J. (2007). Effective top-*k* computation in retrieving structured documents with term-proximity support. In *Proceedings of 16th CIKM conference* (pp. 771–780).

Mingjie Zhu is a PhD student in the department of Electronic Engineering and Information Science at the University of Science and Technology of China. He received his Bachelor's degree from the Special Class for Gifted Young at the University of Science and Technology of China in 2004. Currently he is a visiting student at Microsoft Research Asia.

Shuming Shi is a researcher in Web Search and Mining Group, Microsoft Research Asia. He joined Microsoft in August 2004, just after receiving his PhD degree from Tsinghua University in July 2004. His main research interests are information retrieval and data mining.

Mingjing Li is a researcher in Web Search and Mining Group, Microsoft Research Asia. He is an expert in Chinese handwriting recognition. He received his BS in electrical engineering from the University of Science and Technology of China in 1989, and his PhD from the Institute of Automation, Chinese Academy of Sciences in 1995.

Ji-Rong Wen is currently a research manager in Microsoft Research Asia. He received B.S. and M.S. degrees from the School of Information, Renmin University of China. He received his PhD degree in 1999 from the Institute of Computing Technology, the Chinese Academy of Science. His main research interests are Web data management, information retrieval (especially Web IR), data mining and machine learning.