Optimized Top-K Processing with Global Page Scores on Block-Max Indexes

Dongdong Shan[1] Shuai Ding[2] Jing He[1] Hongfei Yan[1] Xiaoming Li[1]
Peking University, Beijing, China[1]
Polytechnic Institute of NYU, Brooklyn, New York, USA[2]
{sdd, hejing, yhf }@net.pku.edu.cn[1] sding@cis.poly.edu[2] lxm@pku.edu.cn[1]

ABSTRACT
Large web search engines are facing formidable performance challenges because they have to process thousands of queries per second on tens of billions of documents, within interactive response time. Among many others, Top-k query processing (also called early termination or dynamic pruning) is an important class of optimization techniques that can improve the search efficiency and achieve faster query processing by avoiding the scoring of documents that are unlikely to be in the top results. One recent technique is using Block-Max index. In the Block-Max index, the posting lists are organized as blocks and the maximum score for each block is stored to improve the query efficiency.

Although query processing speedup is achieved with Block-Max index, the ranking function for the Top-k results is the term-based approach. It is well known that documents' static scores are also important for a good ranking function. In this paper, we show that the performance of the state-of-the-art algorithms with the Block-Max index is degraded when the static score is added in the ranking function. Then we study efficient techniques for Top-k query processing in the case where a page's static score is given, such as PageRank, in addition to the term-based approach. In particular, we propose a set of new algorithms based on the WAND and MaxScore with Block-Max index using local score, which outperform the existing ones. Then we propose new techniques to estimate a better score upper bound for each block. We also study the search efficiency on different index structures where the document identifiers are assigned by URL sorting or by static document scores. Experiments on TREC *GOV2* and *ClueWeb09B* show that considerable performance gains are achieved.

Categories and Subject Descriptors
H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval.

General Terms
Algorithms, Performance.

Keywords
top-k query processing, early termination, block-max inverted index.

WSDM'12, February 8-12, 2012, Seattle, Washington, USA.

1. INTRODUCTION
Modern commercial web search engines are facing formidable performance challenges due to the data sizes and query loads. The major engines have to process many thousands of queries per second over tens of billions of documents. To deal with this heavy workload, all the engines employ massively parallel systems consisting of thousands of machines. The significant cost of operating these systems has motivated a lot of research into more efficient query processing mechanism.

One major bottleneck in query processing is the length of the inverted list index structure, which could easily grow to hundreds of MBs or even GBs for common search terms (roughly linear with the size of the collection). Given that the modern search systems need to answer queries within fractions of a second, the naïve way to traverse this basic index structure, which could take hundreds of milliseconds for common terms, is impractical.

This problem has long been recognized by researchers, and has motivated a lot of work on optimization techniques, including parallel computation, caching, index compression and Top-k document retrieval (also called early termination or dynamic pruning). In this paper we focus on early termination techniques [1, 3, 5, 10, 12, 15-17, 22, 23, 30, 31, 33, 34]. In a nutshell, early termination techniques speed up the query processing by avoiding the scoring of the documents which are not likely to be in the Top-k results.

The existing techniques for early termination could be roughly divided into two groups: The first group of techniques [1, 3, 16, 17, 22, 23, 30, 31, 33, 34] usually require the inverted lists organized in a way that the most promising documents could be accessed first. Then after processing a certain amount of the most promising documents, the Top-k documents are returned without processing the rest of the less promising documents. In other words, this group of techniques try to "stop early" compared with the naive traversal of the complete lists. The early stop usually happens when the following condition is met: the minimum score in the current Top-k results is larger than the maximal possible score of the unprocessed documents (and sometimes also the rank order of the current Top-k results will not be changed). Thus, for this group of techniques, the way to organize the inverted lists plays a vital role. It decides the number of documents to be processed. Also for this group of techniques, an accumulator table is used to keep track of the most promising documents.

Instead of trying to "stop early", the other group of techniques [5, 10, 12, 15] organize the inverted index lists by document identifier (docID) sorted order and try to "skip" the less promising documents. In practice, the inverted lists are usually stored in blocks (e.g., a block consists of 64 or 128 postings), so each block could be compressed and decompressed separately. To access each single block without decompressing the whole list, one skipping pointer is stored separately for each block, in the form of <*d*,

p>, which indicates that the offset for the particular block is *p* and the minimal (or maximal) docID of that block is *d*. Recent works [12] propose to use the Block-Max index to speed up the query processing. Similar idea also appears in [10]. In the Block-Max index, the maximum score information (e.g., maximum BM25 score) for each block is also stored in the skipping pointer. It is shown in [10, 12] that using block level maximum information would enable us to estimate the score upper bound for one particular document much more accurately, and thus lead to a much better skipping behavior in query processing.

It has some advantages to organize the inverted index by a document sorted order. It gives better inverted index compression and easier conjunctive query processing (described later). Moreover, it is shown in [12] that faster query processing could be achieved. However, they only investigated the ranking function for the Top-k results with term-based score function [10, 12], and it is well known that the documents' static scores such as PageRank are also essential for a good ranking function.

In this paper, we study the methods for efficient query processing on Block-Max indexes when the ranking function consists of documents' static scores such as PageRank, in addition to the term-based scores. We show that the state-of-the-art algorithm BMW in [12] has performance degradation when static scores are added into the ranking function. We propose a new version of BMW called LBMW using local maximum score which outperform the existing ones. Beside this, we also modify the MaxScore algorithm with Block-Max index (called BMM) which has comparable performance with BMW. Basically our improved algorithms can estimate the upper bound score more accurate and find a better candidate document to process. In addition, we also study different methods to dynamically estimate a better score upper bound for each block, compared with the naive way to store both the maximum term-based score and the maximum static score for each block. Essentially, we store the maximum term-based score plus static score, which makes the estimated document upper bound score more accurately. We also empirically study the search efficiency on different index structures where the document identifiers are assigned by URL sorting or by static document scores. The experiments show that our algorithms, combined with our improved score upper bound estimation technique; give the best query processing performance.

2. BACKGROUND

2.1 Index Organization

Modern search engines usually use the inverted index structure to support fast query processing. According to the way of the document identifiers are assigned, there are mainly two types of index organization:

Document-sorted index: In the *document-sorted* index [2, 35], the postings in each inverted list are stored in increasing order of the document identifiers (docIDs). This allows the consecutive docIDs to be stored as *d-gaps* (e.g., d_i is stored as d_i-d_{i-1}-1). With the *d-gap* operation, the list of numbers are changed into a list of smaller gaps and thus we can get a much smaller compressed index, since most of the compression algorithms will benefit from smaller numbers.

Frequency-sorted or **impact-sorted index**: In the frequency-sorted index [17], each inverted list is ordered in decreasing order of the term's frequencies in its corresponding documents (TF). In the impact-sorted index [1, 3] the postings are ordered according to their impact values. Impact value is one small integer indicating the overall contribution of one term to the document's relevance score. In both arrangements, one index list consists of a sequence of groups, within which the postings are equally weighted. Within each group the postings are sorted by docIDs, and then again stored as *d-gaps* to achieve better compression ratio.

With the *d-gaps* representation of the docIDs, the most efficient way to restore the docIDs is to do a sequential scan. In practical, we hope that the inverted list can be accessed randomly to support fast query processing (e.g., conjunctive query). Therefore, the inverted lists are usually divided into blocks [32] (e.g., a block consists of 64 or 128 postings). Usually within each posting list, one skipping pointer is stored for each block at the beginning of the list [6, 10, 12, 35].

2.2 Query Processing

Many query processing methods [1, 5, 10, 16, 17, 22, 23, 27, 29, 35] have been proposed in the IR and Web search areas. Roughly they can be divided into three categories:

Document-at-a-time (DAAT): In the DAAT query processing [4, 5, 10, 12, 16, 35], all relevant inverted lists are simultaneously processed, and the score of a document is fully computed before moving to the next document.

Term-at-a-time (TAAT): In the TAAT query processing [27, 35], only one inverted list is processed at any given time, and the document scores are partial computed in a term-by-term manner. The partial document scores are usually stored in an accumulator table. The complete score of a document is unknown until all query terms are processed.

Score-at-a-time (SAAT): SAAT method [1, 3, 22] is only applicable for the impact-sorted index. It processes one impact block at a time. This method also requires an accumulator table to store documents' partial scores.

For a large collection or for term-dependent queries (e.g. phrase query) [5], DAAT strategies have two advantages compared with TAAT and SAAT methods: (1) DAAT only requires a small-size priority queue to store current Top-k (e.g., 10) results while TAAT or DAAT needs a much larger accumulator table, which is usually expensive to maintain. (2) DAAT methods can easily identify whether query terms satisfy some given conditions in a document (e.g., phrase). In contrast, TAAT and SAAT strategies need keep the occurrences of the first term within the document in the intermediate results, in order to verify whether the second term satisfies the constraint. Please refer to [1, 5, 35] for more detailed comparison among those strategies.

2.3 Pruning Methods

Index pruning is a popular technique for efficient query processing. It can be divided into two categories: **static pruning** and **dynamic pruning** (also called Top-k processing or early termination). For **static pruning** methods [6, 9], the goal is to predict and discard the less important information during the offline index construction, so that the size of the pruned index is much smaller than original full index. The pruned index is usually small enough to be fit into the main memory. Note that static pruning methods usually boost the search efficiency at the cost of search result quality. On the other hand, the **dynamic pruning** methods speed up the query processing by avoiding scoring all documents in the inverted lists during the online search phase.

Various dynamic pruning methods are used for different query processing modes. Those methods could be roughly divided into two groups. One is "stop early", which often requires the inverted list to be organized in such a way that the most promising documents can be evaluated first. The other is "skip", which adapts the skipping pointers into inverted lists and uses some strategies to skip less important documents. Usually, "stop early" methods [1, 3, 16, 17, 22, 23, 30, 31, 33, 34] are used in TAAT or SAAT que-

ry processing mode, while "skip" methods [5, 10, 12, 15] are used in DAAT mode. Another dimension for categorizing the dynamic pruning methods is its safety, indicating whether the results are in the exactly same order as the results from the exhaustive query processing. In this paper, we focus on the safe pruning techniques in DAAT query processing. Furthermore, we focus on disjunctive (OR) queries, which does not disqualify one document when it misses some of the search terms.

2.4 State-of-the-art DAAT pruning methods

As mentioned above, DAAT has many advantages compared with TAAT or SAAT and is shown to be faster in [12] with the Block-Max index structure. Also in the major commercial search engines DAAT is more adapted [5]. Thus in this paper, we focus on query processing with DAAT index traversal. There are two basic state-of-the-art dynamic pruning strategy with DAAT, namely WAND strategy [5, 12, 25] and MaxScore strategy [15, 23, 25, 27].

The original description of the MaxScore [27] strategy does not contain enough details, and it is different from a later implementation by Strohman [23]. Jonassen and Bratsberg [15] presented a more detailed MaxScore algorithm which combines the advantage of both Strohman's and Turtle's implementations. In the MaxScore strategy, it sorts the inverted lists by theirs maximum term scores in the decreasing order before query processing. And during the query processing phase this order is always fixed. After scoring some documents, the current minimum score of the Top-k results is known and it is used as a threshold. With this threshold, the MaxScore algorithm calculates a required term set *T* based on the sorted inverted lists. Thus, any document in the final Top-k results has to contain at least one term in the set *T*. Therefore, we only need to evaluate the documents containing terms in the set *T*. For the terms not in the required set *T*, we only need to move the pointer of their corresponding posting list to the current document candidate chosen from the posting lists in set *T*. In other words, we never use the document candidates from the non-required set *T* to "drive" the query processing. Besides, we can terminate the processing of one particular document when its partial score plus the upper bound score of terms yet to be scored is less than the threshold. Figure 1 gives an example of the MaxScore algorithm. Assume that we are given a query with terms "T_1, T_2, T_3", and their current docIDs are 12, 11 and 5 respectively. Recall that we will store the maximum scores for each list, as 5, 3 and 2 respectively in this example. The current Top-k threshold is 4. So we will know that the required term set *T* is {T_1, T_2}, because the score of a document containing term T_3 only would not exceed the threshold. The MaxScore method selects the current minimum docID 11 in T_2 to calculate its full score. In this case, for term T3, it will skip to docID 11, so the documents with the docID 5 and 10 would not be scored.

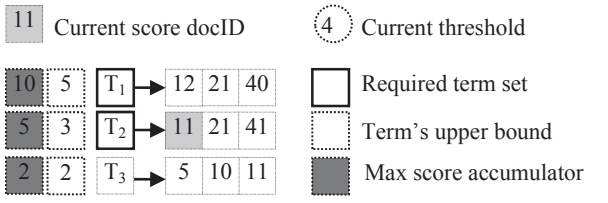

Figure 1. A scenario of processing a 3-term query, where the current pointers in the lists point to docID 12, 11 and 5 (note that lists are sorted by theirs upper bound scores). The MaxScore strategy calculates required term set as T1 and T2, and selects minimum docID 11 from required set. T_3 will skip to docID 11.

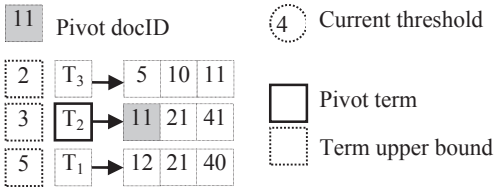

Figure 2. A scenario of processing a 3-term query, where the current pointers in the lists point to docID 5, 11 and 12 (note that lists are sorted by their current docIDs). The WAND strategy will select T_2 as the pivot term and 11 as the pivot docID. If the first term's current docID is identical to the pivot docID, it will calculate a full score for the pivot docID. Otherwise, it moves all earlier posting lists' pointer to the pivot docID 11. In this case, T_3 will skip to docID 11.

In contrast to MaxScore, the WAND [5, 25] strategy sorts the inverted lists by their current docIDs ascending during the query processing phase, which means the order of the posting lists are not fixed. It needs to calculate a pivot term, with the property that the accumulated term's maximum scores from the first term to the pivot term is the first one with a larger score than the threshold. Then it uses the pivot term's current docID as the pivot docID, and compares it with the first terms' current docIDs. If the docIDs are identical, the pivot document is fully scored. Otherwise, it moves all earlier posting lists' pointer to the pivot docID, achieving skipping within these lists. Figure 2 gives a scenario during the processing of query "T1, T2, T3". The reader may refer to the more detail example in the appendix in [25].

2.5 Ranking Function

In this work, we assume that the ranking function consists of: query-independent features, e.g., PageRank, and query-dependent features, e.g., BM25. A similar ranking function has been used in [16, 31]. For the query-dependent IR score, we are not considering the features such as term proximity [7]. Efficient query processing with term proximity will be left for future work. So the ranking function is in the following form:

$$S(d,q) = \alpha \cdot G(d) + \beta \cdot IR(d,q) \qquad (2.1)$$

where *S(d, q)* is the overall score for the document *d* with respect to the query *q*, *G(d)* is the global or static score of *d*, *IR(d, q)* is the query-dependent IR score for the document *d* with respect to the query *q*, α and β are parameters satisfying $\alpha + \beta = 1$. Static score and IR scores are usually normalized into a specific range, e.g., [0, 1] [19, 28]. Craswell et al. [11] proposes another way to combine the static score and IR score. They use a sigmoid function to translate static score then combine IR score without normalization. In this paper, we use the normalized version which is shown to give the best search quality [21, 33]. Our technique could also be applied on Craswell's method and many others.

As shown later, when the above ranking function is used, the state-of-the-art pruning methods will face performance degradation. The reason is that both of the WAND algorithm and the MaxScore algorithm use the maximum IR score (*IR*) and global score (*G*) to select the candidate documents to evaluate. And it is very likely that a posting list contains a document with a high *G* or with a large *IR*, thus the upper bound score of the inverted list would be very large. Could we use the Block-Max index to solve this problem? Does simply storing the maximum IR and maximum *G(d)* for each block give the best performance? Will different document ID assignment techniques play a role? The rest of the paper will focus on these questions.

2.6 Pruning Strategies with Static Scores

When a ranking function has static scores, usually the inverted index is organized in the following ways:

Single segment sorted by static scores [16, 31, 33]**:** When organized in this way, the document identifier is reassigned by their static score, and the promising documents with high static scores are placed at the beginning of the inverted lists. As a result, it is possible to stop early since we will evaluate the documents with high static score first.

Multi-segment divided by impact score [16, 33]: In this structure, the inverted list is divided into multiple segments by one specific kind of term impact scores (e.g., Impact value, term IR score, etc.), and in each segment, the document is sorted by static score or by docID. For one document, its information corresponding to different query terms may appear in different segments. So we still need document score accumulators to maintain the document partial scores

3. OUR CONTRIBUITION

In this paper, we focus on the efficient Top-k document retrieval where the ranking function consists of both documents' static scores and IR scores. We make the following contributions:

(1) We study the performance of the state-of-the-art DAAT strategies (WAND and MaxScore) on the Block-MAX index, considering both the IR score and the static score.
(2) We propose several modified algorithms based on MaxScore and WAND, which make the Top-k document retrieval on the Block-Max index more efficient.
(3) We propose a method to combine the static score and the IR score into one score. We then show how to dynamically use the combined score to give a correct and better estimation of the document's upper bound score.
(4) We evaluate our techniques on the TREC GOV2 and ClueWeb09B datasets. The results show that our algorithms outperform the state-of-the-art techniques.

4. RELATED WORK

We now briefly summarize some previous work. Note that early termination techniques differ in terms of their assumptions about result quality. In this work we only focus on safe (or reliable) early termination techniques that return exactly the same Top-k results in same order as the baseline. Based on the different ranking techniques, previous work on early termination (ET) can be divided into four sets.

In the first set, it is assumed that the ranking function only consists of the IR scores [1, 5, 13, 17, 27], e.g., BM25. In this set there has been a lot of work in both database and IR community. In the database literatures, most of the works assume that the posting lists are sorted by the term scores, and the basic operation in the query processing includes the sequential access and random access. Then optimized algorithms are proposed to minimize the cost, which is measured by the number of sequential access and random access. See [13, 14] for more details. In the IR literatures, some [1, 17] assume that the posting lists are sorted by impact or frequency, and then SAAT index traversal is used to speed up the query processing with the idea of stopping early. Whereas the others assume the posting lists are sorted by docIDs, and usually the maximum IR scores for the posting lists are stored to skip the less promising postings. See [1, 5, 15, 22, 27] for more details. Recently the Block-Max index is used to speed up query processing. In the Block-Max index, the posting lists are sorted by docIDs and the maximum IR score of each block is stored in the skipping pointer. The BMW algorithm is used in [12], and they find it is much faster than the state-of-the-art algorithms SC in [22] when processing disjunctive queries. A block interval based method is used in [10]. It splits blocks into intervals with aligned boundaries, and then it discards the intervals which cannot contain any top results.

In the second set, it is assumed that the ranking function consists of both IR scores and documents' static scores. Long et al. first study this problem in [16] and propose to use the "fancy list" to achieve the goal of stopping early. Fancy list is a 2-segment index, which contains the document with high IR score in fancy part, and in each segment, the docID is sorted by static score. Zhang et al. [31] propose some methods to organize inverted list by the global information combines the static score and the upper bound of its IR scores for all terms contained in document.

In the third set, the term proximity scores are considered in the ranking function [25, 30, 34]. Usually, an auxiliary structure such as phrase index or term pair's index is employed to calculate the term proximity scores. Then it uses the method of the second set to evaluate the documents.

In the fourth set, hundreds of features are considered in the ranking function which learned from machine learning methods. In practice, it is not possible to score every document using expensive machine learned ranking function. Thus, there are several works to solve the efficient problem in this field, such as multi-stages ranking [8], cascade ranking model [28] and so on. Usually, those methods view retrieval as a multi-stage progressive refinement problem. Each stage uses a more complex ranking model in a smaller candidate document set. Ours describe methods below can be used as a first stage ranking model. Note that those methods are not safe pruning techniques.

Independent of what is mentioned above, another idea in IR query processing, namely, docID reordering, is also relevant to our work. Generally, there are two kinds of docID reordering methods: one is to minimize the index size and the other is to optimize the dynamic query pruning. In the first case, a number of recent works [19, 20] show that inverted index size can be reduced a lot by reordering the docIDs. Intuitively, if we assign similar pages with close docIDs, the *d-gaps* between the consecutive documents in the inverted lists are much smaller, and the required space for the *d-gaps* is reduced. Moreover, as shown in [12, 29], docID reordering in this way can significantly increases the speed of processing both conjunctive and disjunctive queries. The most commonly used method for reordering docIDs in this way is to sort the documents by their URLs, and then assign the docIDs sequentially. In the second case, documents are sorted descending by their static scores, and then the docIDs are assigned sequentially. This reordering technique would place the documents with larger static scores at the beginning of inverted lists, so the important documents can be scored first, and then it is more likely that we will stop the query processing when the early termination condition is reached. It has been shown by experiments [16, 33, 34] that the static score ordered index do not perform well in early termination when the weight of static score in the ranking function is assigned small, e.g., α =0.2. Our work here shows that it can enhance the query processing speed a lot based on Block-Max index even when the weight of the static score is small.

5. OUR ALGORITHM

Query processing on the Block-Max index mainly has three steps:
Step 1: Select a candidate document to be scored. In this step, both WAND and MaxScore strategies can be used.
Step 2: Estimate the upper bound score for the candidate document using the block maximum information.

Step 3: Score the document if its estimated upper bound score is larger than the current top-k threshold score, otherwise select the next candidate document.

There are three important problems in this framework:

1) What is the best way to find a candidate document?
2) How the upper bound score of a candidate document can be estimated accurately?
3) When shall we score a document, and how can we terminate earlier after acquiring some partial score information?

5.1 Ranking Function

Our ranking function is based on the formula (2.1) discussed in Section 2. We use PageRank as the document static score *G(d)*, and normalize it with formula (5.1) [21, 33]:

$$G_{PR\_norm} = \frac{\log(1 + G_{PR})}{\log(1 + G_{\max\_PR})} \tag{5.1}$$

We use BM25 formula [6, 18] to calculate the IR score, and we set parameter k_2 and k_3 to zero for convenience [6, 21].

$$S_{bm25} = \sum_{t \in q} w^i \cdot \frac{tf_{(t,d)} \cdot (k_1 + 1)}{tf_{(t,d)} + K} \tag{5.2}$$

where w^i is the query term i's IDF value. $tf_{(t,d)}$ is term frequency in document d. Maximum BM25 score can be calculated when $tf_{(t,d)}$ is set large enough [21, 33] . The maximum BM25 score is:

$$S_{\max\_bm25} = \sum_{t \in q} w^t \cdot (k_1 + 1) \tag{5.3}$$

We can calculate a normalized BM25 score by formula 5.4:

$$S_{bm25\_norm} = \frac{S_{bm25}}{S_{\max\_bm25}} \tag{5.4}$$

In [16] the BM25 score is normalized using the local maximum score in the inverted lists as S_{bm25_norm}, and all our techniques could be applied using that normalization.

5.2 Local Block-Max WAND (LBMW)

The basic BMW algorithm is proposed in [12]. Our LBMW algorithm improves it in the pivot term selecting phase. In our method, we use the local block upper bound score to select the pivot term while the BMW algorithm still uses the global term upper bound score as in [5]. Usually, the local block upper bound score is much smaller than the global one. Thus, our pivot term selection method can select a better pivot term. Figure 3 shows an example on LBMW to select the pivot term. In this example, each block consists of two docIDs. The maximum IR score of each block is represented as the superscript in "()". After sorting lists by their current docIDs, we know that T_1's current docID (90) is the current maximum docID. The LBMW method calculates a local block maximum score for each term from their current docID to the current maximum docID (90). In this scenario, T_3, T_2 and T_1's local maximum scores are 1.3, 2.5 and 3.0 respectively. So we can calculate the pivot term in LBMW is T_1, because the cumulated maximum scores for T_3(1.3) and T_2 (3.8) are both smaller than the threshold score 4. Thus the pivot docID in LBMW is 90. In the BMW, it uses global term maximum score 2(T_3), 2.8(T_2) and 3.0(T_1) to calculate the pivot term. So the pivot term in BMW is T_2 (T_2' cumulated score is 4.8) and the pivot docID is 61. So the next processing document in BMW method is 61, but its score actually cannot exceed the threshold 4. The processing on this document wastes of time. In our LBMW method, docID 61 will be skipped.

Local block upper bound score can be calculated very efficiently by recording last calculated score and its block's maximum docID. For example, T_3's list iterator can keep a temp local maximum score 1.3 and its corresponding maximum docID 111 to avoid redundant calculation. Similarly, we can also calculate its local maximum static score by this method. This idea is similar to the interval-based method in [10]. The interval-based method splits the blocks with aligned boundaries before query processing, while the LBMW method aligns document boundaries during query processing.

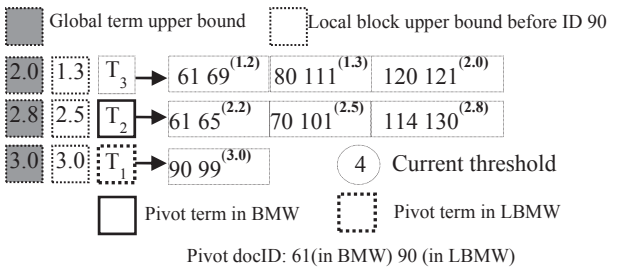

Figure 3. LBMW sorts lists by their current docIDs, and then calculates the local block maximum for each term from their current docID to current maximum docID 90. In this scenario, T_3, T_2 and T_1's local maximum scores are 1.3, 2.5, and 3.0. Then it uses these local maximum scores to calculate pivot term. In this case, pivot term in LBMW is T_1, while pivot term in BMW is T_2.

5.3 Local Block-Max MaxScore (LBMM)

The MaxScore algorithm in [15, 23, 27] can be easily extended on the Block-Max index, by adding an estimating step for the upper bound score, before scoring the candidate document. We call this extended method Block-Max MaxScore (BMM). The candidate document selecting step in BMM is similar to the example in Figure 1. It selects a minimum docID in the required term set as its candidate document. In BMM, we add an estimating step before scoring document. With this step, we can calculate document's upper bound score based on the block maximum information. If the score surpasses the threshold, we score this candidate document; otherwise, we will select the next validated candidate document.

Scoring document in MaxScore is slightly different from that in WAND. In the MaxScore (as shown in Figure 1), the inverted lists have been sorted by their maximum IR score, so the important term is scored first. While in WAND, the inverted lists are sorted by their current docID, and it is not likely that the important term would be scored first. When scoring a document, we can estimate the document's upper bound by summing its current score and the maximum scores of the unseen terms. In the MaxScore, the unseen terms upper bound score can be computed once before the query processing starts. In our implement, we use the local block upper bound score to estimate the unseen terms' upper bound score instead of the term's global upper bound score as in [15]. Usually, the local maximum score is much smaller than the global score. So we can estimate document's upper bound score more accurately. We call this method as Local Block-Max MaxScore (LBMM).

5.4 Estimate Upper Bounds More Accurately

One important operation in LBMW or LBMM algorithms is to estimate the upper bound score of the candidate document. One

natural way is to store the maximum normalized PageRank score and the maximum IR score respectively in the skipping table. Then we can use those two maximums to estimate the overall upper bound score. Intuitively, this method is better than using global maximum static score and global maximum term IR score to estimate the upper bound score since the local upper bounds are much closer to the real scores than the global upper bounds. In the LBMW and LBMM, we use this method to estimate the upper bound score. However, in one block, the document with the maximum term IR score and the document with the maximum static score usually are usually not identical. Thus, the overall upper bound would be overestimated by respective maximum term IR score and maximum static score. Is there a better way to estimate the upper bound more accurately?

Our answer is yes. The main idea is to combine the static score and term IR score, and use the combined score as the block's single maximum score. But it is a little complex to combine those two scores to give the correct results, due to the normalization used in the ranking function. First, we can calculate a combined score for each document in a block by Formula 5.5. Then we can select the maximum combined score as the block maximum score.

$$\begin{aligned} S(d,q) &= \alpha \cdot G(d) + \beta \cdot IR(d,q) \\ &= \alpha \cdot G_{norm}(d) + \beta \cdot \frac{\sum_{t \in q} w^t \cdot \frac{tf_{(t,d)} \cdot (k_1+1)}{tf_{(t,d)}+K}}{\sum_{t \in q} w^t \cdot (k_1+1)} \\ &= \frac{\alpha \cdot G_{norm}(d) \cdot \sum_{t \in q} w^t + \beta \cdot \sum_{t \in q} w^t \cdot \frac{tf_{(t,d)}}{tf_{(t,d)}+K}}{\sum_{t \in q} w^t} \\ &= \frac{\sum_{t \in q} (\alpha \cdot G_{norm}(d) \cdot w^t + \beta \times w^t \cdot \frac{tf_{(t,d)}}{tf_{(t,d)}+K})}{\sum_{t \in q} w^t} \end{aligned} \quad (5.5)$$

In Formula 5.5, we note that the static score needs to multiply a maximum term score when combining with term IR score. Using this combined score, it will underestimate document's upper bound score in two cases:

1. When one list for term T_i ends up and other lists still have candidate documents. In this case, the static upper bound score of candidate documents may be underestimated, because the static scores assigned to inverted list of term T_i have not been considered.
2. One of term's block maximum combined score is lower than the current candidate document's static score. Figure 4 shows an example in this case. In the figure, each block has two docIDs, and the maximum combined score is represented as the superscript in "()". Each document's term IR score and normalized PR score is represented as the superscript in order in "<>". We assume T_1 and T_2's IDF value (w^{T1}, w^{T2}) are equal to 10 and α=0.3. According to formula 5.5, we can get T_1's and T_2' first block maximum combined score is 4.8 and 1.0 respectively. When using this maximum combined score to estimate a document's upper bound, it can get the upper bound score as (4.8+1.0)/(10+10)=0.29. But the document with ID 20's real score is 0.9*0.3+3*0.7/(10+10)=0.375. So this document's upper bound score is underestimated. This case happens when one document's static score is much larger than the block's combined score, here document with ID 20's static score is 0.9*0.3=0.27, T_2's first block's maximum combined score is 0.1*0.3+1.0*0.7/(10+10) = 0.1.

We have known that the underestimation happens when the document static score is larger than the other block's combined score. Thus, we need to store the maximum static score in blocks to check whether this situation actually happens. If this happens, we can use the maximum static score to estimate the upper bound score, otherwise, we will use the combined score. To the first case, we can use the similarly method.

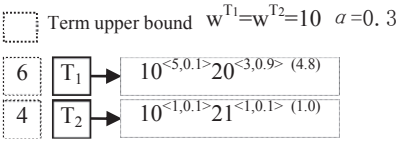

Figure 4. Document's upper bound score is underestimated in this example if only using the maximum combined score.

6. EXPERIMENTS

6.1 Experimental Setup

Datasets: We test our algorithms using two TREC collections, as illustrated in Table 1. The GOV2 collection consists of 25.2 million Web pages crawled from the .gov Internet domain in 2004 and it consumes 426GB of disk space uncompressed. The ClueWeb09B collection is a subset of the ClueWeb09 collection which consists of roughly the first 50 millions of English Web pages and it consumes 1.5TB of disk space uncompressed. We preprocess the documents by removing the html tags and index the pages' content, URL and anchor text. Each token is stemmed by a porter stemmer. For the inverted list whose length is longer than 512, we group 64 postings into one block, and use PForDelta algorithm [32] to compress its docIDs and term frequencies respectively. The compressed indexes for the GOV2 and ClueWeb09B collection consume 10.6GB and 23.4GB respectively. For each block, we can store block maximum BM25 score, maximum PageRank score, and the maximum combined version of PageRank and BM25 score. But for different dynamic pruning algorithm, we build different indexes which only store two of those scores. These scores are represented by float (4 byte) uncompressed, and each score takes about 313MB (2.8%) for the GOV2 index and 0.7GB (3%) for the ClueWeb09B index.

Table 1. The statistic information for GOV2 and ClueWeb09B index

	Documents	Text	Index size
GOV2	25M	426G	10.6G
ClueWeb09B	50M	1.5T	23.4G

Query Sets: We use **TREC 2006 Efficiency 10k Queries (GOV2#10K)** and **TREC 2009 Million Query 40k Queries (MQ09#40K)** as our test query sets for the GOV2 and ClueWeb09B datasets respectively. We use a list of 400 stopwords provided by Lemur toolkit to remove the stop words (we only remove stop words in the queries. In the inverted index, it still contains those stopwords.). Figure 5 shows the distributions of query lengths in two query sets. Note that the lengths reported here don't contain the stopwords or the terms not appearing in the collection. The average query length in GOV2#10K and MQ09#40K is 3.47 and 2.29 respectively. The average posting number of GOV2#10K and MQ09#40K queries is 5.37M and 6.56M respectively. We run all the experiments on the Dell R905 server with four Quad-Core AMD 8380 (2.5GHz) processors (but we only use one core of them) and 64GB of RAM. We load the full index into the main memory before processing the queries. Without specifying, we return top-10 results for each query.

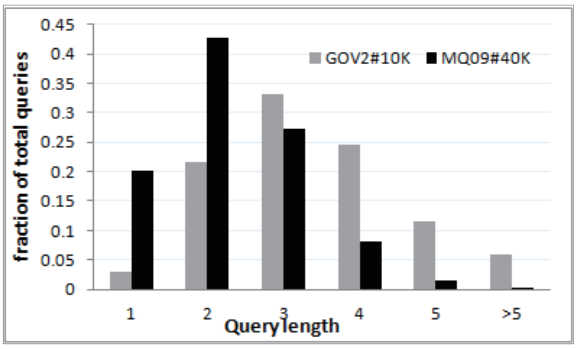

Figure 5. Query length distrubute in GOV2#10K and MQ09#40K

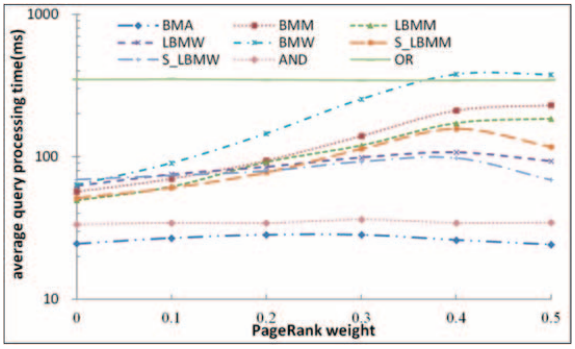

Figure 6. Average query processing time for different static score weights on the random ordered GOV2 dataset.

Index structures and pruning strategies: We compare different pruning strategies on three index organizations using DAAT query processing. Note that our pruning strategies are safe pruning methods which mean the results returned by our methods are exactly same as exhaustive query processing.

Exhaustive AND (OR): Exhaustive conjunctive or disjunctive query processing [2].

BMW: The WAND pruning strategy on Block-Max index [12] .

BMM: The MaxScore pruning strategy on Block-Max index.

LBMW: An improved version of BMW which uses the algorithm in section 5.2 to find a better pivot document in selecting document step.

LBMM: An improved version of BMM, which uses the algorithm in section 5.3 to estimate the documents' upper bound in scoring document step.

S_LBMW and S_LBMM: An improved version of LBMW and LBMM. They use the algorithm in section 5.4 to give a better document's upper bound estimation in estimating document upper bound step.

BMA: Query processed for conjunctive queries on the Block-Max index.

6.2 Basic Results

In this section, we list the performance of various pruning strategies on the index where document identifies are randomly assigned (we first randomize the documents in the dataset, and then assign docIDs sequentially.).

In the Figure 6, there are four main observations: Firstly, when the weight of PageRank (PR) is small, most of pruning strategies perform well, especially for the MaxScore based algorithms; e.g., when the weight of PR is zero, BMM and LBMM achieve 12% and 22% improvement compared with BMW and LBMW respectively. The performance of BMW drops rapidly when the weight of PR increases, and it performs even worse than naïve OR when PR's weight is large than 0.4. The main reason is that BMW (or BMM) still uses global term maximum IR score and global maximum PR score to select candidate documents. Secondly, the methods with the local maximum information (LBMM, LBMW) perform better than those using global information. One reason is that with the local maximum IR and PR score, LBMW can find a better candidate documents than BMW which uses the global maximum IR and PR score. For the LBMW, with local block maximum IR score, it can estimate the document's upper bound more accurately and for LBMM, it can terminate earlier when scoring a document. Thirdly, the methods (S_LBMM, S_LBMW) using maximum IR+PR to estimate the documents' upper bound scores also improve the query processing efficiency. Finally, we note that all the pruning methods still have a performance gap compared with BMA and Exhaustive AND on GOV2 Dataset. Though BMA and AND methods are much faster than the other pruning methods, they will lead to inferior search quality. Since relevant documents do not contain all the query words are ignored by those methods. We get similarly results on the ClueWeb09B dataset (Due to the limited of space, the results are not shown.). One interesting observation in the ClueWeb09B dataset is that all the pruning methods are better than the exhaustive AND operation when the weight of PageRank is 0. The S_LBMW method performs better than the exhaustive AND operation in a wide range of PR weight setting.

Table 2 shows the average query processing time with different query lengths. We set the weight of static score is 0.2 as in [26]. Firstly, we can find that for the queries containing more than four terms, the MaxScore based strategies are better than the WAND based strategies. The main reason is that the MaxScore based methods select candidate documents in the required term set which is likely to be composed of important terms, while the WAND based methods select candidate documents based on its docIDs without considering the terms' importance. Secondly, for the queries having fewer terms, the LBMW based methods are performing better. In this case, the LBMW and S_LBMW will use the local block maximum IR and PR scores to calculate a candidate documents. In LBMM based methods, it still uses global term maximum score and maximum PR score to select a candidate document. So the candidate document selected by LBMW based methods is likely to be better than the one selected by MaxScore based methods. We also note that when the query length is two, the pruning strategies are faster than exhaustive conjunctive (AND) query processing.

Table 2. Average query processing time in ms for different number of query terms (DataSet:GOV2, Index Type: random α=0.2)

	AVG	1	2	3	4	5	>5
OR	347	35	140	288	418	577	843
BMM	94.0	3.1	27.2	68	118	173	276
BMW	146	2.5	36.1	96	182	284	475
LBMM	91.7	3.6	27.1	69	117	165	249
LBMW	84.9	2.5	30.2	63	101	151	257
S_LBMM	**77.5**	3.0	**19.2**	56	100	**144**	**229**
S_LBMW	79.5	**2.5**	20.8	**53**	**97**	152	270
AND	34.2	34	30.6	35	36	35	33
BMA	29.1	2.8	15.8	29	36	39	37

Table 3 shows some detail statistic information for each phase of the query processing when the static score weight is set as 0.2. From Table 3, we can see that all the pruning strategies can improve the performance greatly compared to the exhaustive OR

query processing. The S_LBMW only needs to score 1.1% of number of documents processed by exhaustive disjunctive (OR) operation and its number of evaluated documents are even much smaller than that of AND query processing. Note that the evaluated documents we report here contain the partial evaluated documents. The BMM based methods evaluate much more documents. One reason is that those methods need to calculate partial document scores to determine whether it can stop scoring the candidate document or not. While the WAND based methods can test if the document needs scoring by comparing the pivot docID to the first term's current docID. If they are not identical, it will skip scoring the document. As a result, the WAND based methods can perform better when the ranking function is expensive. Another interesting finding in Table 3 is that the MaxScore based methods decode fewer integers than WAND based methods. The S_LBMM algorithm only decodes 12% of integers compared with exhaustive OR. The main reason is that MaxScore based methods can skip a lots of blocks in the unimportant terms' inverted lists (usually, they are much longer than important terms' lists). Thus, MaxScore based methods are better if the decoding is the bottleneck. The document upper bound estimation times we report here is the called times for the actually estimation function. We judge if the block in each list is moved to a new block or not before we call the estimation function. If not, the latest estimated score can be used. This trick reduces the estimation time a lot. The average estimation number used by the MaxScore based methods is still fewer than the WAND based methods.

Table 3. The average number of evaluated documents, document upper bound estimation times, decoded integers and decoded blocks for different methods (DataSet:GOV2, α=0.2).

	evaluated docs (K)	estimation times (K)	decoded ints (M)	decoded blocks (K)
OR	4248.2	0	10.51	84.1
BMM	582.9	66.7	1.48	15.5
BMW	210.6	109.7	6.86	63.6
LBMM	582.9	65.4	1.25	13.8
LBMW	55.9	90.4	5.16	56.5
S_LBMM	538.4	**65.4**	**1.23**	**13.8**
S_LBMW	**48.1**	85.6	4.58	50.9
AND	53.0	0	3.28	31.1
BMA	19.9	40.2	2.61	25.5

6.3 DocID Reassignment by URL Sorting

In this section, we show the impact of document reassignment on Top-k processing. The work [12, 19, 20, 24] show that reassigning docIDs by their similarities can not only improve index compression ratio but also speed up the conjunctive and disjunctive query processing. One most commonly used method to assign docID according to the content similarities is to sort the documents in the URL order.

Figure 7 show the performance of different dynamic pruning methods on GOV2 dataset with docID reassignment strategy. We can find that the average query processing time is reduced a lot compared with random ordered index in the Figure 6. The average processing time almost decreases by half when the weight of PR is small. This docID reassignment strategy makes the documents in one block similar to each other, so the maximum score in one block is closer to the scores of the average documents. Thus it helps to skip more unimportant blocks in the inverted list. We also find that the query processing performance is improved for the exhaustive OR operation, mainly due to smaller storage size of the inverted index and thus the better CPU caching performance. One interesting finding is the LBMW and S_LBMW are faster than

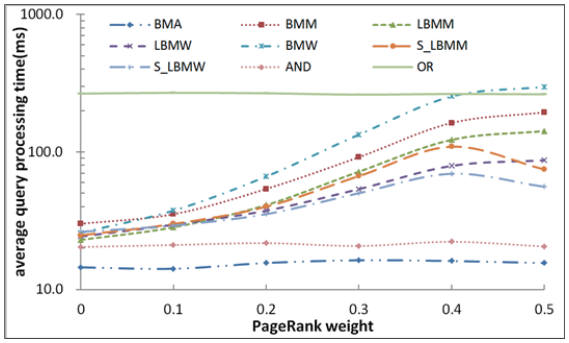

Figure 7. Average query processing time for different static score weights on the URL ordered GOV2 dataset

LBMM and S_LBMM in URL ordered index. As shown in Figure 7, the S_LBMM is better than LBMW and S_LBMW when static score weight is less than 0.2. Thus, the WAND based methods can get more benefit when the scores in one block become smoother. We also get similarly results in the ClueWeb09B dataset. One interesting finding is the ClueWeb09B dataset is the speed of S_LBMW is even faster than exhaustive AND operation in the Clueweb09B dataset. One reason is that relevant documents processed by exhaustive AND operation in the Clueweb09B dataset is about five times more than that in the GOV2 dataset. Thus, the exhaustive AND operation needs to evaluate more documents. In Table 5, we find that the number of documents evaluated by the S_LBMW is 11% of that by the exhaustive AND operation, and the decoded integers are almost half of the later.

Table 4. The average number of full evaluated docIDs, condition checked times, decoded integers and decoded blocks for different methods on URL ordered index (Dataset: GOV2, α =0.2).

	evaluated docs (K)	estimation times (K)	decoded ints (M)	decoded blocks (K)
OR	4248.2	0	10.51	84.1
BMM	498.0	36.3	1.07	12.3
BMW	186.3	47.5	3.24	29.4
LBMM	498.0	35.8	0.92	11.8
LBMW	57.0	34.0	2.19	22.7
S_LBMM	432.0	35.8	**0.91**	**11.8**
S_LBMW	**52.0**	**33.1**	1.92	20.2
AND	53.0	0	1.55	13.9
BMA	9.8	16.0	1.0	10.4

Table 5. The average number of full evaluated docIDs, condition checked times, decoded integers and decoded blocks for different methods on URL ordered index (Dataset: ClueWeb09B, α=0.2).

	evaluated docs (K)	estimation times (K)	decoded ints (M)	decoded blocks (K)
OR	5901.0	0.0	13.1	105.0
BMM	388.1	61.3	1.4	15.7
BMW	125.4	81.7	4.0	40.4
LBMM	388.1	60.1	1.1	14.2
LBMW	41.7	72.3	3.2	36.7
S_LBMM	268.7	**60.0**	**1.1**	**14.2**
S_LBMW	**28.1**	64.6	2.1	26.2
AND	262.4	0.0	5.1	46.9
BMA	20.7	32.4	2.3	25.1

From Table 4, we note that the documents' upper bound estimation time is reduced by almost a half compared with that of the

random ordered index. One reason is that after document reassignment, the candidate documents are more likely to be in the same blocks. So we can get more documents with high score, if decoding one block. Thus the number of decoded integers can be reduced a lot (as shown in Table 4 and Table5). One interesting finding is that though the ClueWeb09B dataset contains double of documents of the GOV2, the number of the retrieved documents by the exhaustive OR operation only increases by 38.9% and the retrieved documents by the exhaustive AND operation almost increases five times. One reason is that the query of MQ09#40K is much shorter than those of GOV2#10K, as shown in Figure 4. Besides, the BMA method can reduce the number of evaluated documents and decoded much less integers compared with exhaustive AND, and it is the fastest processing method.

6.4 DocID Reassignment by Static Score

Figure 8 show the performance of different pruning strategies on the GOV2 dataset with PageRank ordered. In this kind of index organization, the "stop early" method can be combined to achieve fast query processing. We also run the pruning OR [16, 33, 34] which stops scoring document if it finds the maximum score of unprocessed document is smaller than current threshold. It takes about 320ms to process query when the PR weight is less than 0.3 on the GOV2 dataset. When the PR weight is larger, it performs better. It takes 265ms and 125ms to process one query when the PR weight is set as 0.4 and 0.5. Pruning OR improves about 3 times compared with the exhaustive OR when PageRank weight is set to 0.5. But things change when using the Block-Max index. We find that as the PageRank weight increases, the performance of different pruning strategies all improved a lot. It is very different from URL ordered index, whose performances decrease when PR weight increases. One interesting finding is that the MaxScore based methods are faster than WAND based methods when PR weight is less than 0.4. The main reason is that MaxScore based methods can get more benefit from PR ordered index in selecting a candidate document (In URL or Random order index, it still uses global maximum PR value to select a candidate document. In PR order index, it can use current maximum PR value). The BMW method is the fastest one when PR weight equal to 0.5. It takes only 9ms to process a query, almost 13.9 times faster compared with the Pruning OR 125ms in this case.

From the Figure 8, we find that all the pruning methods are faster than the exhaustive AND operation. One reason is that the MQ09#40K is much shorter and it has more relevant documents in the ClueWeb09B dataset (about five times of those in GOV2) when processing conjunctive queries. Another reason is that the pruning methods perform better when the length of query is shorter.

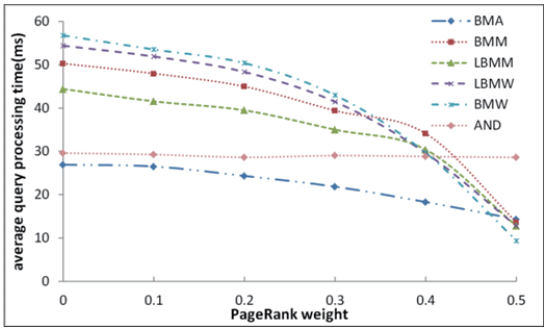

Figure 8. Average query processing time for different static score weights on the PageRank ordered GOV2 dataset

Table 6. The average number of full evaluated docIDs, condition checked times, decoded integers and decoded blocks for different methods on PageRank ordered index(Dataset: GOV2, α=0.2).

	evaluated docs (K)	estimation times (K)	decoded ints (M)	decoded blocks (K)
OR	4248.2	0.0	10.51	84.1
BMM	155.2	41.7	0.90	9.4
BMW	13.9	57.6	3.05	35.3
LBMM	155.2	**40.8**	**0.75**	**8.2**
LBMW	**10.2**	56.0	2.89	34.6
AND	53.0	0.0	2.73	25.7
BMA	7.0	31.5	1.48	15.4

From Table6, we find that the average evaluated documents is reduced more than three times for MaxScore based methods and five time for WAND based methods compared with that in URL ordered index. For the LBMM method, the number of decoded integers is also reduced by 18% compared with that in URL ordered index. The BMA method still is the fastest query processing method.

6.5 Increase K of the Top-K Processing

In this section, we show the query performance as we increase the parameter K. Figure 9 shows the experimental results on the GOV2 with URL ordered index and PR weight is set to be 0.2. As shown in the chart, the performance for the exhaustive AND operation and OR operation are quite stable as they decode and evaluate all the documents for any K. Other methods generally need more time for getting more top documents. We find that the processing time is in the scale of $\log_{10}$K. As K becomes larger, the performance gap between S_LBMM and S_LBMW becomes smaller and they both outperform the other pruning methods. The situation almost same in the Clueweb09B dataset or in other organized indexes, those results are not shown for the space constraint.

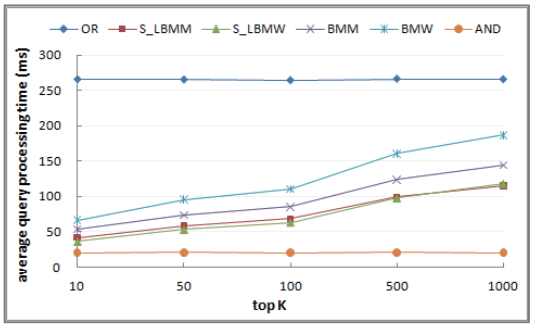

Figure 9. Average query processing time as increasing K (Dataset: GOV2, α=0.2, Index type: URL ordered)

7. CONCLUSION

To conclude, our query processing framework using Block-Max index contains three steps: 1) Select a candidate document; 2) Estimate its upper bound score; 3) Score the document if its upper bound score is larger than the threshold. Based on this framework, we study efficient query processing techniques when the ranking function consists of both the IR scores and documents' static scores. In particular, we propose the BMM pruning strategy, and it performs as well as the state-of-the-art method BMW. We also propose the LBMW and LBMM for selecting better candidate documents. Experiments show that the performances of the local based methods (LBMW, LBMM) are better than global based

methods (BMW, BMM). We have studied how to dynamically estimate a better overall upper bound score for each block. We make efforts to give both correct and faster results using this improved upper bound estimation. Besides, we also study the search efficiency on different index structures where the document identifiers are assigned by URL sorting or by the static document scores. We show experimentally that our algorithms (S_LBMW and S_LBMM), combined with our improved score upper bound estimation technique, can give the best query processing performance.

Acknowledgements

This work is supported by NSFC Grant (60933004, 70903008, 61073082) and HGJ Grant 2011ZX01042-001-001. We would like to thank Torsten Suel and the anonymous reviewers for their helpful comments.

8. REFERENCES

[1] Anh, V.N. and Moffat, A. 2006. Pruned query evaluation using pre-computed impacts. *SIRGIR'06* (New York, NY, USA, Aug. 2006), 372-379.

[2] Anh, V.N. and Moffat, A. 2006. Structured Index Organizations for High-Throughput Text Querying. *SPIRE* (2006), 304-315.

[3] Anh, V.N., de Kretser, O. and Moffat, A. 2001. Vector-space ranking with effective early termination. *SIGIR'01* (New York, NY, USA, Sep. 2001), 35-42.

[4] BRIN, S. and PAGE, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*. 30, 1-7 (Apr. 1998), 107-117.

[5] Broder, A.Z., Carmel, D., Herscovici, M., Soffer, A. and Zien, J. 2003. Efficient query evaluation using a two-level retrieval process. *CIKM'03* (New York, NY, USA, 2003), 426-434.

[6] Büttcher, S. and Clarke, C.L.A. 2006. A document-centric approach to static index pruning in text retrieval systems. *CIKM'06* (New York, NY, USA, 2006), 182-189.

[7] Büttcher, S., Clarke, C.L.A. and Lushman, B. 2006. Term proximity scoring for ad-hoc retrieval on very large text collections. *SIGIR'06* (2006), 621-622.

[8] Cambazoglu, B.B., Zaragoza, H., Chapelle, O., Chen, J., Liao, C., Zheng, Z. and Degenhardt, J. 2010. Early exit optimizations for additive machine learned ranking systems.*WSDM'10* (New York, NY, USA, 2010), 411-420.

[9] Carmel, D., Cohen, D., Fagin, R., Farchi, E., Herscovici, M., Maarek, Y.S. and Soffer, A. 2001. Static index pruning for information retrieval systems. *SIGIR'01* (New York, NY, USA, 2001), 43-50.

[10] Chakrabarti, K., Chaudhuri, S. and Ganti, V. 2011. Interval-based pruning for top-k processing over compressed lists. *ICDE'2011*, 709-720.

[11] Craswell, N., Robertson, S., Zaragoza, H. and Taylor, M. 2005. Relevance weighting for query independent evidence. *SIGIR'05* (New York, NY, USA, 2005), 416-423.

[12] Ding, S. and Suel, T. 2011. Faster top-k document retrieval using block-max indexes. *SIGIR'11* (New York, NY, USA, 2011), 993-1002.

[13] Fagin, R. 2003. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*. 66, 4 (2003), 614-656.

[14] Ilyas, I.F., Beskales, G. and Soliman, M.A. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.* 40, 4 (2008), 11:1--11:58.

[15] Jonassen, S. and Bratsberg, S. 2011. Efficient Compressed Inverted Index Skipping for Disjunctive Text-Queries. *Advances in Information Retrieval*. (2011), 530-542.

[16] Long, X. and Suel, T. 2003. Optimized query execution in large search engines with global page ordering. *VLDB'03* (Sep. 2003), 129-140.

[17] Persin, M., Zobel, J. and Sacks-Davis, R. 1996. Filtered Document Retrieval with Frequency-Sorted Indexes. *JASIS*. 47, 10 (1996), 749-764.

[18] Robertson, S.E., Walker, S. and Hancock-Beaulieu, M. 1998. Okapi at TREC-7: Automatic Ad Hoc, Filtering, VLC and Interactive. *TREC* (1998), 199-210.

[19] Shieh, W. 2003. Inverted file compression through document identifier reassignment. *Information Processing & Management*. 39, 1 (Jan. 2003), 117-131.

[20] Silvestri, F. 2007. Sorting out the document identifier assignment problem. *Advances in Information Retrieval*. (Apr. 2007), 101-112.

[21] Song, R., Wen, J.R., Shi, S., Xin, G., Liu, T.Y., Qin, T., Zheng, X., Zhang, J., Xue, G. and Ma, W.Y. 2004. Microsoft research asia at web track and terabyte track of trec 2004. *TREC'04* (2004).

[22] Strohman, T. and Croft, W.B. 2007. Efficient document retrieval in main memory. *SIGIR'07* (New York, NY, USA, 2007), 175-182.

[23] Strohman, T., Turtle, H. and Croft, W.B. 2005. Optimization strategies for complex queries. *SIGIR'05* (New York, NY, USA, 2005), 219-225.

[24] Tonellotto, N., Macdonald, C. and Ounis, I. 2011. Effect of different docid orderings on dynamic pruning retrieval strategies. *SIGIR'11* (New York, NY, USA, 2011), 1179-1180.

[25] Tonellotto, N., Macdonald, C. and Ounis, I. 2010. Efficient dynamic pruning with proximity support. *Large-Scale Distributed Systems for Information Retrieval* (2010), 33-37.

[26] Tsai, M.-F., Liu, T.-Y., Qin, T., Chen, H.-H. and Ma, W.-Y. 2007. FRank: a ranking method with fidelity loss.*SIGIR'07* (New York, NY, USA, 2007), 383-390.

[27] Turtle, H. and Flood, J. 1995. Query evaluation: strategies and optimizations. *Inf. Process. Manage.* 31, 6 (1995), 831-850.

[28] Wang, L., Lin, J. and Metzler, D. 2011. A cascade ranking model for efficient ranked retrieval. *SIGIR'11* (New York, NY, USA, 2011), 105-114.

[29] Yan, H., Ding, S. and Suel, T. 2009. Inverted index compression and query processing with optimized document ordering. *WWW'09* (2009), 401-410.

[30] Yan, H., Shi, S., Zhang, F., Suel, T. and Wen, J.-R. 2010. Efficient term proximity search with term-pair indexes. *CIKM'10* (New York, NY, USA, Oct. 2010), 1229-1238.

[31] Zhang, F., Shi, S., Yan, H. and Wen, J.-R. 2010. Revisiting globally sorted indexes for efficient document retrieval. *WSDM'10* (New York, NY, USA, Feb. 2010), 371-380.

[32] Zhang, J., Long, X. and Suel, T. 2008. Performance of compressed inverted list caching in search engines. *WWW'08* (New York, NY, USA, 2008), 387-396.

[33] Zhu, M., Shi, S., Li, M. and Wen, J.-R. 2007. Effective top-k computation in retrieving structured documents with term-proximity support. *CIKM'07* (New York, NY, USA, 2007), 771-780.

[34] Zhu, M., Shi, S., Yu, N. and Wen, J.-R. 2008. Can phrase indexing help to process non-phrase queries? *CIKM'08* (2008), 679-688.

[35] Zobel, J. and Moffat, A. 2006. Inverted files for text search engines. *ACM Comput. Surv.* 38, 2 (2006)