Document Prioritization for Scalable Query Processing

Hao Wu
University of Delaware
Newark, DE, USA
haow@udel.edu

Hui Fang
University of Delaware
Newark, DE, USA
hfang@udel.edu

ABSTRACT

Query latency is an important performance measure of any search engines because it directly affects search users' satisfaction. The key challenge is how to efficiently retrieve top-K ranked results for a query. Current search engines process queries in either the conjunctive or disjunctive modes. However, there is still a large performance gap between these two modes since the conjunctive mode is more efficient with lower search accuracy while the disjunctive mode is more effective but requires more time to process the queries.

In this paper, we propose a novel query evaluation method that aims to achieve a better balance between the efficiency and effectiveness of top-K query processing. The basic idea is to prioritize candidate documents based on the number of the matched query terms in the documents as well as the importance of the matched terms. We propose a simple priority function and then discuss how to implement the idea based on a decision tree. Experimental results over both Web and Twitter collections show that the proposed method is able to narrow the performance gap with the conjunctive and disjunctive modes when K is larger or the length of a query is longer. In particular, compared with one of the fastest existing query processing methods, the propose method can achieve a speedup of 2 with marginal loss in the retrieval effectiveness on the Web collection.

Categories and Subject Descriptors: H.3.4 [Systems and Software]: Performance evaluation (efficiency and effectiveness)

General Terms: Performance, Experimentation

Keywords: query evaluation; document prioritization; efficiency

1. INTRODUCTION

Efficiency is an important performance measure of large-scale Information Retrieval (IR) systems. Despite the increasing amount of online information, users have come to expect the IR systems to return high quality search results with a short response time, usually under a second for each query. It has been shown empirically that query latency has a direct impact on search users' satisfaction as well as search engine revenues [5,7,28]. Thus, it is critical to improve query processing efficiency and minimize query latency.

Given a query, IR systems need to rank documents based on their relevance scores, which are often computed based on the occurrences of query terms. A straightforward query processing method is to traverse the postings of all the query terms in the inverted index and fully compute the relevance scores of all the documents in the postings, i.e., those containing at least one query term. Such a *disjunctive* query processing mode (i.e., OR mode) is able to accurately generate the ranking list for a given retrieval function. However, with the rapid growth of the collection size, this basic disjunctive mode becomes impractical, since its processing time increases linearly with the number of documents in the collection while users often expect sub-second response time.

One possible way of improving the efficiency is to use a *conjunctive* (i.e., AND) mode instead [3, 31, 32]. It first identifies the documents that match all query terms and then computes their relevance scores. Conjunctive mode is clearly more efficient since fewer documents need to be evaluated. However, it could hinder the effectiveness of search engines since many relevant documents may not contain *all* the query terms.

An alternative solution is to improve the disjunctive mode through dynamic pruning techniques [6,12,16–18,27,29,33]. Instead of exhaustively evaluating every document that matches at least one query term, dynamic pruning techniques evaluate only a smaller set of documents that have greater potential to make to top-K results and bypass the evaluation for the rest of the documents. These techniques are often rank-safe for top-K results, which means that they can generate the same top-K results as the exhaustive evaluation of the disjunctive mode. Thus, dynamic pruning can improve the query processing efficiency without hurting the effectiveness. Unfortunately, despite the previous efforts, there is still a large performance gap between disjunctive and conjunctive query processing modes, especially when the number of retrieved results, i.e., K, or the number of terms in a query is large.

In this paper, we present a novel and scalable query processing method based on document prioritization. The basic idea is to reduce the number of documents that need to be fully evaluated by prioritizing documents based on the number of matched query terms as well as their importance. To

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM'14, November 3–7, 2014, Shanghai, China.
Copyright 2014 ACM 978-1-4503-2598-1/14/11 ...$15.00.
http://dx.doi.org/10.1145/2661829.2661914 .

implement this idea, we build a decision tree to quickly classify documents into different blocks based on the matched query terms, and these blocks, together with the documents in the blocks, are then ranked based on the importance of the matched query terms. On one hand, the proposed method includes more documents in the evaluation process than the conjunctive mode, which leads to more effective results. On the other hand, it uses a simple strategy to prioritize all the documents matching at least one query term so that the number of documents that need to be fully evaluated is smaller than the disjunctive mode, which means more efficient query processing.

We conduct experiments to evaluate the performance of the propose method in two search domains, i.e., Web search and microblog search. In terms of the efficiency, the proposed method can reduce the query processing time by a factor of 2 for longer queries or larger K on Web collections, compared with the state of the art query processing technique, i.e., block-max index methods [16, 18]. It can also consistently improve the efficiency even for smaller K on Twitter collections, but the speed up is not as big as on the Web collections. In terms of the effectiveness, the proposed method is much more effective than the conjunctive mode, and it is only slightly worse than the disjunctive mode on the Web collections and can generate even more accurate search results on the Twitter collections.

2. BACKGROUND

With the increasing amount of online information and the rapid growing number of queries, commercial search engines have leveraged many performance optimization techniques, including parallel computing [5], caching [4], index compression [1,10,36,38] and dynamic pruning [6,12,16,18,27,29,33], to meet strict performance constraints - high throughput and low latency. We now provide some background about a few most related topics to our study.

2.1 Basic Query Processing

When processing a query, IR systems need to traverse the inverted lists of all query terms and compute the relevance score of each document with respect to the query based on a retrieval function. The query processing strategies can be classified into two classes:

- **Term-At-A-Time (TAAT):** It sequentially processes the inverted lists of all query terms and accumulates the partial document scores contributed by each term [8, 23, 39]. Each document requires an accumulator to record the accumulated relevance score.
- **Document-At-A-Time (DAAT):** It processes the inverted lists of all query terms in parallel. A document needs to be fully evaluated based on the contribution of all query terms that occur in the document before moving to the next one [6, 16, 18, 22, 30, 33].

In this paper, we focus on DAAT strategies, since it is more commonly used by commercial search engines and can work well in both disjunctive and conjunctive modes [6, 30].

Note that these basic query processing techniques can be considered as a *disjunctive* mode since we consider only documents in the inverted lists of query terms, i.e,. documents that contain at least one query term.

2.2 Top-K Query Processing

Large-scale IR systems need to keep up with the exponential growth of the collection size, and it has become impossible to accurately compute the relevance score for every document in the collection. Indeed, one of the major problems that have been studied is to efficiently find top-K ranked results for a given query based on a retrieval function.

Various dynamic pruning (i.e., early termination) techniques have been proposed for DAAT with the aim of reducing the number of documents that need to be fully evaluated by avoiding the scoring of postings that can not make the top-K results [6, 12, 16–18, 33]. All these methods need to maintain two types of information: (1) a pruning threshold, which records the smallest score of documents that can make to the top-K results; (2) the maxscore of a term, which is the highest impact score of all the postings in the inverted list of the term. For example, the Maxscore method [33] first identifies essential terms based on their maxscores and the threshold, and then skips the scoring of documents that does not contain any essential terms. The WAND method [6] takes a different approach based on pivot terms. Specifically, terms are first sorted based on the current document IDs in their inverted lists, and a pivot term is then selected so that the sum of maxscores of this term as well as all the terms that are ranked in front of it is greater or equal to the pruning threshold. After this, the document ID of the pivot term will then be used to skip documents with smaller IDs. More recently, the block-max index methods [12,16–18] have been proposed to further reduce the query processing time. The basic idea is to store the highest impact score for each block in the block-max indexes and then leverage this information to further improve the performance of the Maxscore and WAND algorithms by skipping more documents

It is worth pointing out that all the pruning methods we have discussed are *rank safe* for top-K results since the pruning results are the same as those generated by exhaustively evaluating every document.

Another way of generating top-K results is to use the *conjunctive* mode, i.e., evaluating only documents that contain all query terms. Previous work has focused on fast posting list intersection [2, 32]. Conjunctive mode is often more efficient than the disjunctive mode with pruning, but it would hurt the effectiveness since it may miss many relevant documents and generate results with lower recall.

2.3 Multi-phase Retrieval Architecture

Learning to rank methods have been shown to be more effective than traditional retrieval functions [21], but they are more computational expensive. To achieve a better balance between the efficiency and effectiveness, commercial search engines often use a two phase retrieval architecture [3,11,32]. In the first phase, a simple and fast retrieval function, such as a linear interpolation of Okapi BM25 and document prior, is used to generate a list of candidate documents that are potentially relevant. In the second phase, a more accurate but computationally expensive ranking function, such as learning to rank methods [21] will be used to re-rank the candidate documents generated in the first phase.

The first phase is essentially top-K query processing. It is clear that the effectiveness of this stage would affect the final ranking results. Intuitively, when K is larger, more potential relevant documents will be included in the second phase, leading to better search accuracy. However, it would

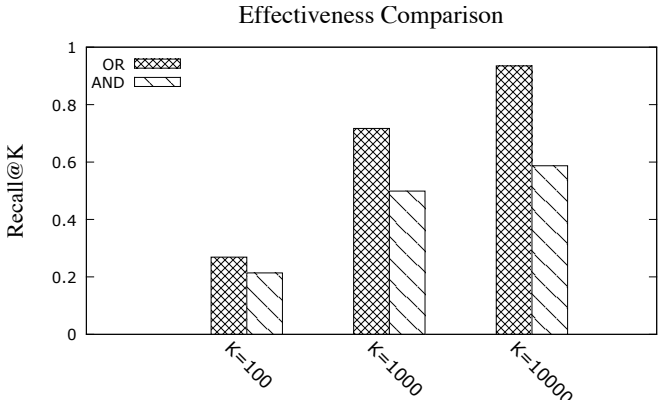


Figure 1: Effectiveness - AND returns fewer relevant documents for all values of K.

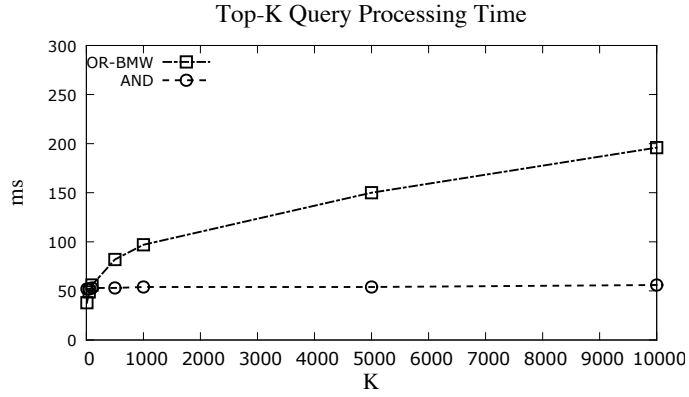


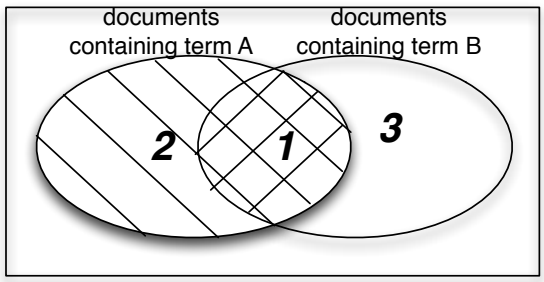
Figure 2: Efficiency - The performance gap becomes wider for larger K.

undoubtedly hurt the efficiency since more documents need to be evaluated. Some previous studies on learning to rank have used a large value for K, such as 1,000 [25], 5,000 [14] and even "tens of thousands" [13]. A more recent study also showed that smaller K can hurt the final search quality and suggested that at least 2,000 documents should be used to ensure maximum effectiveness [22]. Clearly, it is critical to study how to scale query processing techniques for larger values of K.

3. DOCUMENT PRIORITIZATION

3.1 Motivation

Although the conjunctive mode is more efficient, it could hurt the effectiveness since many relevant documents may not contain all the query terms. As shown in Figure 1, the recall values of the conjunctive method (AND) are consistently lower than the disjunctive method (OR) for all the retrieval cut-offs. It means that many relevant documents will be missed in this step. In the multi-phase retrieval architecture described earlier, the final retrieval performance would be affected significantly because the second phase only re-ranks the results of the first phase and is unable to bring additional relevant documents.



Conjunctive mode: consider only documents from 1
Disjunctive mode: consider documents from 1 and 2 and 3
Our proposed method (*Priority*): Assume IDF(A)>IDF(B).
We select documents from 1 first, and then 2, and then 3.

Figure 3: Difference between conjunctive mode, disjunctive mode and the proposed method.

Significant efforts have focused on dynamic pruning methods to improve the query processing time in the *disjunctive* mode [6, 18, 33]. These techniques generally perform better when K (the number of retrieved results) is smaller, but can not scale well. Figure 2 shows the average query processing time of the conjunctive method and one of the fastest query processing strategy (i.e., BMW [18]) in disjunctive mode for different values of K. It is clear that BMW does not scale well since the performance gap between this method and AND becomes larger when the value of K increases.

Thus, it would be interesting to study how to further narrow the performance gap when K is larger without hurting the effectiveness as much as the conjunctive method.

3.2 Basic Idea

Query processing time is closely related to the number of documents that need to be fully evaluated for a given retrieval function, such as Okapi BM25 [26]. The *disjunctive* mode without pruning evaluates all documents matching at least one query term. All these documents are treated *equally*, and their relevance scores with respect to the retrieval function are then fully computed.

Intuitively, not every document passing the Boolean OR filter has the same likelihood of being relevant. The basic idea of document prioritization is to prioritize documents based on an approximation of the relevance scores and then compute the accurate relevance scores for the top-K highly prioritized documents instead of accurately computing the relevance scores for all documents. Clearly, the key challenge is how to *efficiently* and *effectively* prioritize the documents.

Dynamic pruning methods in the disjunctive mode can be regarded as one specific method of document prioritization. They try to prioritize documents based on their fully or partially computed relevance scores for the given retrieval function. They can produce the exact top-K results, but are not very efficient due to the fact that the relevance scores are costly to compute. Conjunctive mode can be regarded as another specific method of document prioritization. It essentially splits all documents passing the Boolean OR filter into two categories, i.e., whether they contain all query terms or not, and focuses only on the documents passing the Boolean AND filter. This method is efficient but not very effective.

To overcome the limitations of existing methods, we propose a new way of document prioritization with the goal

of using a simple yet effective retrieval function to *approximate* the results generated by the given retrieval function. It is well known that the retrieval effectiveness is related to the use of multiple retrieval signals such as TF (term frequency) and IDF (inverse document frequency) [19]. Existing retrieval functions such as Okapi BM25 often combine these retrieval signals in a complicated way, which requires the traversal of inverted lists of all query terms and the combination of statistics obtained from the lists. A recent study discussed the potential of applying individual simple signals sequentially for effective retrieval [35]. Their results suggest that it could lead to comparable or even more effective results if we rank documents first using a single strong signal such as IDF and then use other signals or ranking methods to break the ties (i.e., re-rank the documents with the same scores computed using the previous signals).

Motivated by the previous study [35], we propose to prioritize documents based on the sum of the IDF values of all distinct query terms that occur in the documents. Formally, the priority score of document D for query Q is computed as follows:

$$S_P(Q, D) = \sum_{t \in Q \cap D} IDF(t), \quad (1)$$

where t is a query term. $IDF(t)$ is the inverse document frequency of t and can be computed as $\log \frac{N+1}{df(t)}$, where N is the number of documents in the collection and $df(t)$ is the number of documents containing term t. It is clear that the documents are prioritized based on the number of *distinct* query terms that they contain and the importance of the terms. Figure 3 shows an example scenario with a two-term query. If the IDF value of A is larger, documents from the area 1 (i.e., those with both terms) would have higher priority than those from area 2 (i.e., documents with only term A), which have higher priority than those from area 3 (i.e., documents with only term B).

This function is chosen to strike a better balance between the efficiency and effectiveness. First, IDF has been shown to be the stronger retrieval signal than others such as TF, since it has higher upper bound performance [35]. Second, the number of distinct values of the priority scores is small, which makes it possible to efficiently process documents using the tree-structure that will be described in Section 3.3.

After the document are prioritized based on Equation (1), we can then take at least K documents with higher priorities and re-rank them with more accurate relevance scores computed based on the given retrieval function. We will explain how to efficiently implement this idea in Section 3.3 and discuss its efficiency and effectiveness in Section 3.4.

3.3 Tree-based Prioritization

We have explained the basic idea of document prioritization for query processing in the previous subsection. We now explain how to implement it efficiently based on a tree-based structure.

Recall that the basic idea is to prioritize documents based on the number of distinct query terms that they contain and the importance of the matched terms. Thus, if a query Q contains $|Q|$ terms, the priority scores computed from Equation (1) would have only $2^{|Q|}$ different values. Since a collection may contain millions or even billions of documents, it means that many documents would have the same priority scores.

This observation motivates us to use a decision-tree based data structure that can quickly classify documents into different blocks based on the matched query terms and then prioritize the document blocks (i.e., documents with the same priority score) accordingly. One advantage of the tree-based implementation is its ability to exploit the relations among different blocks, which can improve the processing time when the number of blocks is large (as shown in Section 4.5.3).

Constructing a decision tree: Given a query, the decision tree can be constructed as follows. All query terms are first ranked based on their IDF values, and their ranks determine which levels of the decision tree they will correspond to. We put terms with higher IDFs at the higher levels because this enables the fast pruning of nodes corresponding to the non-essential terms, which will be discussed soon. Each non-leaf node will have two children. One child includes the documents containing the corresponding term, while the other child includes the documents that do not contain the corresponding term. Each node is associated with a priority score, which is determined by the priority score of the documents belonging to the node. Each leaf node contains a document block, in which all the documents have the same priority score as computed using Equation (1). Figure 4 shows an example decision tree for a query with two terms A and B. Since $IDF(A)$ is larger than $IDF(B)$, the nodes at the first level corresponds to A and those at the second level correspond to B. Node "A" in the first level has a priority score of $IDF(A)$, and "not A" has a priority score of 0. Moreover, the first leaf node has a priority score of $IDF(A) + IDF(B)$ since this node corresponds to the documents matching both terms.

Tree-based query processing: After building a decision tree for a query, we traverse the indexes in a similar way as DAAT. The inverted lists of all query terms are processed in parallel, and the documents are processed in the order of their document IDs. For each document, instead of directly computing its priority score, we leverage the decision tree and put the document into the corresponding block. The priority of a document is the same as that of its corresponding block, which is computed when building the tree. The last block (the one with the lowest priority score) will always be empty since it corresponds to the documents that do not match any query terms and these documents would not occur in the inverted lists of query terms. As shown in Figure 4, when processing $d1$, we would put it to the first block based on the constructed decision tree. After processing all the documents, we can then return all the documents in the first M blocks so that they are the smallest set of blocks that can cover K documents. For example, if $K = 3$, we would return the first two blocks as shown in the Figure 4. When putting a document into a block, the term statistic information related to the document will also be stored in the decision tree. After identifying the top M blocks, we will then fully compute the relevance scores of all the documents in the blocks based on the given retrieval function (e.g., Okapi BM25) and return top-K results.

Dynamic pruning: Since we focus on top-K query processing, it would be more efficient if we could skip some documents that will not make the final top-K results. We propose a dynamic pruning strategy that can be combined with the above tree-based query processing. After placing a document into a block of the decision tree, we will check

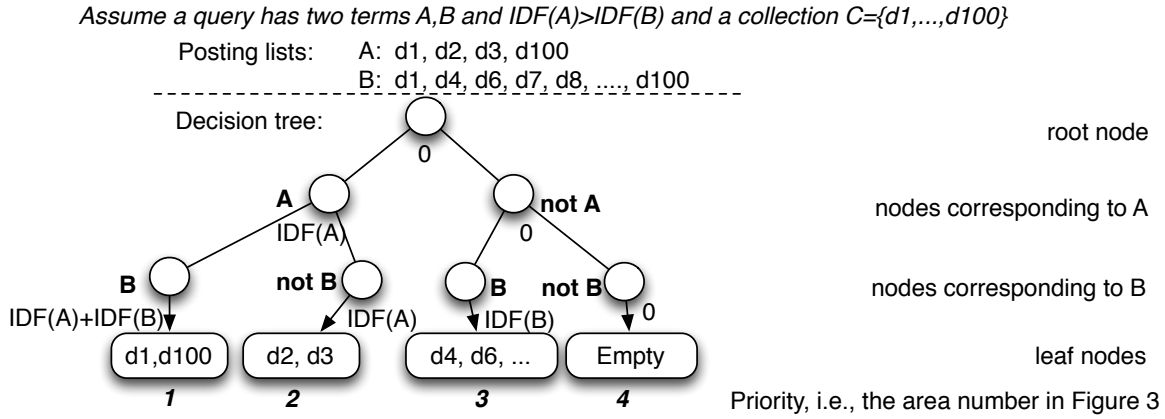


Figure 4: An example of the decision tree based query processing

whether this block together with the blocks with higher priority are big enough to cover top-K documents. If so, we will disable all the nodes with blocks that have lower priority. If all the children of a node are disabled, this node will also be disabled. If a disable node corresponds to the occurrence of a term, we can refer to this term as a *non-essential* term. All the documents that contain only non-essential terms will not make the final top-K results. Thus, for the inverted lists of non-essential terms, we can then skip the documents that do not contain any essential terms by moving their current document pointers to the smallest document IDs in the inverted lists of the essential terms, i.e., those corresponding nodes in the tree are not disabled. Note that this idea is similar to the Maxscore method [33] since we also try to exclude non-essential terms from the query processing.

Let us go back to the example shown in Figure 4. Assume we focus on retrieving top 2 results. According to DAAT, $d1$ is the first one to be processed, and it will be put into the first block since it contains both query terms. $d2$ and $d3$ are the next, and will be put into the second block. Since $K = 2$ and the first two blocks already cover 3 documents, we can now disable all the remaining blocks and associated nodes, i.e., all the nodes from the sub-tree of "not A". It means that B is a non-essential term since the documents containing only document B will not make to the final top-K results. Based on our dynamic pruning method, the current document ID in the inverted list of B will be moved to $d100$ since it is the smallest document ID in the inverted list of the essential term, i.e., term A. After processing $d100$, we can put it into the first block and return only the documents from the first block. Finally, the two documents from the first block will be evaluated with the given retrieval function such as Okapi BM25. It is clear that the dynamic pruning makes it possible to skip many documents that can not make to the top-K results, i.e., $d4$, $d6$, etc.

3.4 Discussions

Efficiency: The efficiency of any query processing method is closely related to the number of documents that need to be fully evaluated with the given retrieval function. The main advantage of our approach is to first prioritize the documents using an efficient tree-based query processing technique to reduce the number of documents that need to be evaluated with the given retrieval function. In particular, the number of evaluated documents in the proposed approach is determined by the size of the first M blocks. When the size of the first M blocks is smaller than the number of documents that need to be fully evaluated by existing query processing methods, the proposed method is expected to be more efficient. Since the number of evaluated documents in any query processing method would grow with the value of K and the block size in the proposed decision tree would shrink when a query has more terms, we expect that the proposed method could improve the efficiency better when K is larger or the queries are longer.

Effectiveness: The proposed method is similar to the conjunctive mode since both of them try to use a simple method to select a small set of documents that need to be evaluated. However, our method is more effective since it can include more potentially relevant documents than in the conjunctive mode. Since IDF is a very strong retrieval signal [35], the number of missed relevant documents in our method would be small.

4. EXPERIMENTS

4.1 Data Sets

We compare the proposed method with a couple of the state of the art query processing techniques on two large-scale search domains, i.e., Web search and Microblog search. Due to the high cost associated with creating relevance judgments, there is no single collection that has a large number of queries with relevance judgments. Thus, for each search domain, we have to use different collections for evaluating the effectiveness and efficiency.

Web search: We use the TREC Gov2 collection as the data set. The collection contains 25.2 million web pages crawled from the .gov domain. To evaluate the retrieval effectiveness, we use two sets of 50 queries from the TREC 2005-2006 Terabyte tracks [9], which are denoted as *TB05* and *TB06*. To evaluate the efficiency, we randomly choose 1,000 queries from the efficiency queries used in the same tracks. They are denoted as *TB05L* and *TB06L*.

Microblog search: To evaluate the retrieval effectiveness, we use the TREC Tweets2011 collection as the data set. The collection contains 16 million tweets posted from 01/23/2011 to 02/08/2011. Two query sets from TREC 2011 and 2012 Microblog tracks [24] are used and denoted as *MB11* and *MB12*. To evaluate the efficiency, we use a larger collection, i.e., a subset of the twitter7 [37] collection, as

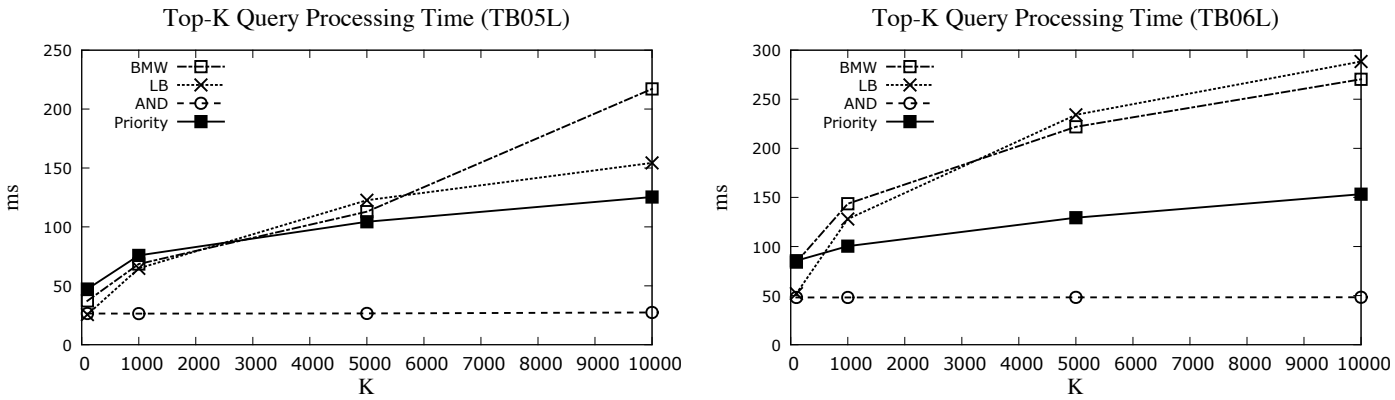


Figure 5: Efficiency comparison on *TB05L* (left) and *TB06L* (right): average processing time per query (ms)

the data set. The data set contains 253 million tweets from 06/07/2009 to 10/21/2009. We crawled trending queries for each date in the corresponding period from Google Trends and randomly pick 1,000 queries to use in the experiments. Although they are not queries submitted to Twitter, they can represent user information needs for the corresponding time period. This collection is denoted as *MBL*.

4.2 Experiment Setup

We now provide more contextual information about our implementations. We use the inverted indexes whose postings in an inverted list are sorted based on document IDs. The indexes are compressed using a state of the art method, i.e,. New-PFD compression method [36]. Following previous studies [18], an inverted list is split to blocks. Each block contains 64 postings, and the blocks can be decompressed separately. Additional information about the block, such as the block size and the uncompressed maximum document ID in the block, is stored, which makes it possible to skip blocks without decompression.

The proposed tree-based document prioritization method with pruning as described in Section 3.3 is denoted as *Priority*. Specifically, to retrieve top-K results, the *Priority* method would first use the proposed prioritization method to identify top ranked blocks covering at least K documents, score the documents from these blocks using the Okapi BM25 method, and then retrieve top-K documents as the final results. The *Priority* method will be compared with three baseline methods: (1) *BMW* [18]: the state of the art dynamic pruning method for disjunctive mode, which has been shown to be more efficient than the WAND [6] and Maxscore methods [33]; (2) *LB* [16]: an improved version of BMW using live blocks, which has been shown to achieve a speed-up of 2 over *BMW* for top-10 query processing; (3) *AND*: the conjunctive mode implemented using the WAND method [3, 6].

All experiments are conducted via a single machine with dual AMD Lisbon Opteron 4122 2.2GHz processors. We use the Okapi BM25 [26] as the retrieval function and retrieve top-K results. We evaluate the retrieval efficiency with the average query latency, i.e., how much time on average it takes to execute a query. We measure the effectiveness with MAP@K and Recall@K.

Table 1: Effectiveness Comparison (Recall@K)

K		100	1,000	10,000
TB05	BMW/LB	0.269	0.717	0.935
	AND	0.214	0.499	0.587
	Priority	0.268	0.676	0.923
TB06	BMW/LB	0.337	0.724	0.906
	AND	0.290	0.512	0.700
	Priority	0.331	0.696	0.901
MB11	BMW/LB	0.412	0.699	0.845
	AND	0.132	0.135	0.135
	Priority	0.430	0.695	0.843
MB12	BMW/LB	0.301	0.662	0.846
	AND	0.142	0.181	0.191
	Priority	0.311	0.670	0.840

4.3 Performance Comparison: Efficiency

The first set of experiments is to compare the average query processing time per query for both search domains. We will provide more result analysis in Section 4.5.

Figure 5 shows the comparison results over different values of K (i.e., the number of retrieved documents) on the two Web collections, i.e., *TB05L* and *TB06L*. Note that the performance of AND varies with the different values of K, but the differences are too small to see in the figures.

There are a few interesting observations. First, the proposed *Priority* method clearly has a constant factor speed improvement over the state of the art dynamic pruning methods (i.e., BMW and LB) on both collections. In particular, the speed up of the *Priority* method is around 2 when K is large. Second, the *Priority* method does not perform as well as BMW when the value of K is smaller. Finally, although LB is able to achieve a speed-up of 2 over BMW when K is smaller, which is consistent with the previous study [16], it does not always have a clear advantage over BMW for larger K.

We also conduct similar experiments for Microblog search. Figure 6 compares the average query processing time of the four methods. It is interesting to see that the *Priority* method is consistently more efficient than the BMW and LB methods for all the retrieval cut-offs, although the speedup is not as large as what we observe in the Web search domain.

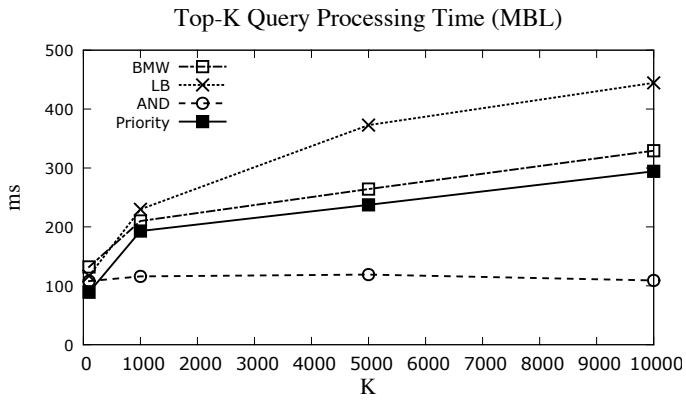


Figure 6: Efficiency comparison for *MBL*: average processing time per query (ms)

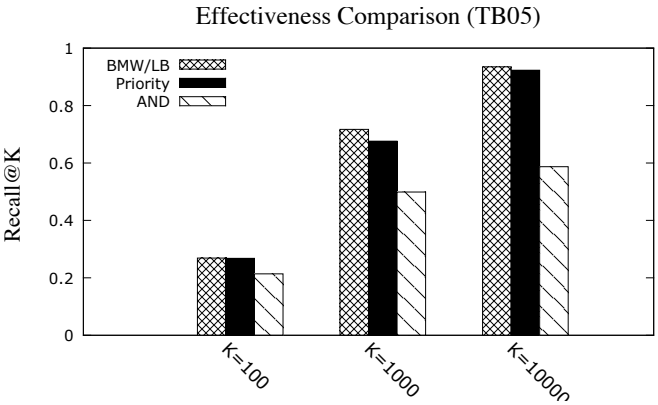


Figure 7: Effective comparison on *TB05*: Recall@K

4.4 Performance Comparison: Effectiveness

The proposed *Priority* method is not rank safe for top-K results, since the priority function we used is a simple approximation of the original retrieval function. Thus, it would be interesting to compare it with the three baseline methods in terms of the retrieval effectiveness. Note that *BMW* and *LB* have the same effectiveness since both of them are rank-safe. Figure 7 shows the recall at different retrieval cut-offs on the *TB05* collection. And we also report the performance measured with recall and MAP in Table 1 and Table 2 for all the collections.

It is clear that the effectiveness of the *Priority* method is slightly worse than that of *BMW* for Web search, but the effectiveness loss is much smaller than that caused by the *AND* method. And such an observation is consistent for different retrieval cut-offs, different effectiveness measures and different collections. Moreover, for the Microblog search, the effectiveness of *Priority* is very similar to that of the *BMW* method on both collections, and both of them are much more effective than the *AND* method. It is interesting to see that the Priority method is more effective than the other two methods. This observation suggests that IDF is a very strong retrieval signal for microblog search since the short length of tweets makes other signals such as TF and document length normalization less effective. This observation is consistent with the previous study on tie-breaking [35].

Table 2: Effectiveness Comparison (MAP@K)

K		100	1,000	10,000
TB05	BMW/LB	0.167	0.324	0.343
	AND	0.132	0.236	0.245
	Priority	0.165	0.306	0.322
TB06	BMW/LB	0.188	0.285	0.293
	AND	0.163	0.247	0.252
	Priority	0.187	0.281	0.290
MB11	BMW/LB	0.206	0.245	0.247
	AND	0.077	0.079	0.079
	Priority	0.221	0.265	0.268
MB12	BMW/LB	0.137	0.193	0.203
	AND	0.086	0.094	0.095
	Priority	0.148	0.208	0.217

4.5 Result Analysis

As shown earlier, the proposed method can strike a better balance between the effectiveness and efficiency than the baseline methods by significantly reducing the query processing time without sacrificing too much on the quality of search results. We now conduct more experiments to better understand the performance differences.

4.5.1 Efficiency comparison for different query lengths

To further understand the performance behavior, we report the average query processing time for different query lengths. Table 3 shows the average query processing time when retrieving top 1,000 documents on the *TB05L* collection. It is interesting to see that the *Priority* method is more efficient than *BMW* when the queries have more than two terms. Moreover, it can achieve a speed-up around 2 when the number of query terms is larger than 4. We can make consistent observations on *TB06L* collection as well, as shown in Table 4. Indeed, the average query length of *TB06L* is larger than that of *TB05L*, which may explain the better speed up achieved by the *Priority* method for smaller Ks as shown in Figure 5.

In summary, the proposed *Priority* method is more scalable than the *BMW* method and it is more efficient for longer queries and larger values of K. This trend can be clearly seen in the Figure 8, which shows the average query processing time for different query lengths and different retrieval cut-offs on *TB06L*. Note that the trend is similar on the other data set and for the other method (i.e., *LB*).

4.5.2 Efficiency: the number of evaluated documents

Here we conduct more analysis to better understand the efficiency comparison results between *Priority* and *BMW*. The comparison with *LB* is similar and ignored due to the space limit. Since the query processing time is closely related to the number of documents that need to be fully evaluated, the performance gain of *Priority* probably comes from its ability to quickly prune many documents with lower priorities.

To verify this hypothesis, we report the number of evaluated documents for *TB06L* as shown in Table 5. It is clear that the query processing time is indeed related to the number of evaluated documents. When more documents need to be fully evaluated, it would take more time for query pro-

Table 3: Avg. processing time per query (ms) for different query length on *TB05L* (K=1,000)

Query length	2	3	4	5	6	> 6
BMW	36.13	75.01	110.27	133.93	184.17	290.37
Priority	90.93	59.62	64.81	56.41	83.48	122.35
Speed-up	0.4	**1.3**	**1.7**	**2.4**	**2.2**	**2.37**

Table 4: Avg. processing time per query (ms) for different query length on *TB06L* (K=1,000)

Query length	2	3	4	5	6	7	> 7
BMW	52.41	107.01	146.54	238.06	344.94	383.38	531.81
Priority	82.01	86.95	93.93	126.85	153.17	206.87	268.07
Speed-up	0.64	**1.2**	**1.6**	**1.9**	**2.3**	**1.9**	**2.0**

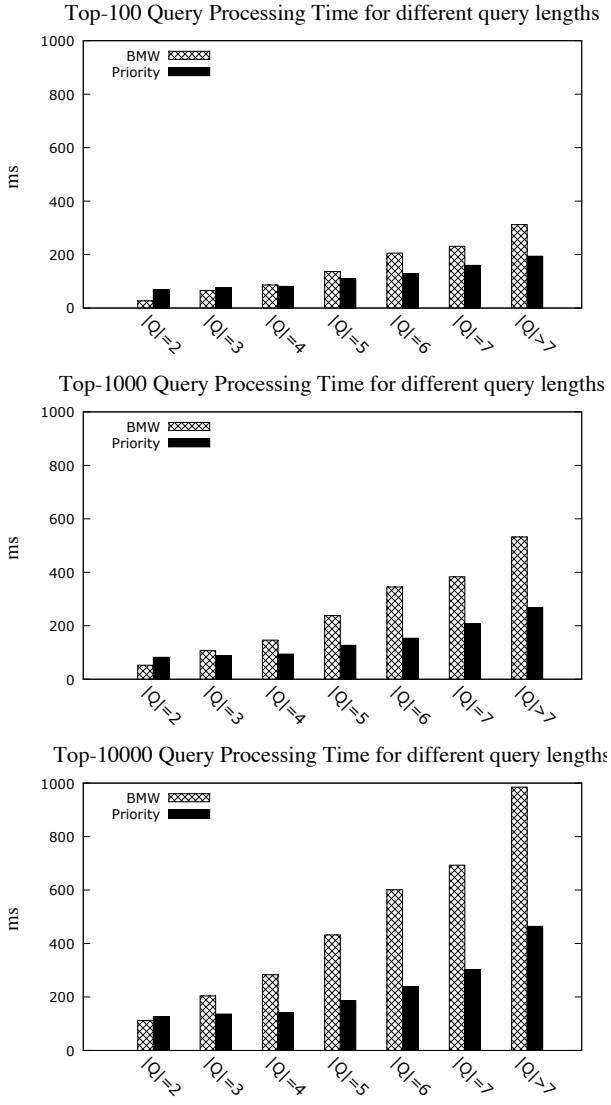


Figure 8: Performance comparison on *TB06*: average query processing term per query (ms)

Table 5: Avg num. of evaluated docs on *TB06L*

K	100	1,000	5,000	10,000
BMW	**35,104**	105,277	246,089	357,357
Priority	66,409	**83,506**	**112,996**	**140,688**

cessing. It is clear that the number of evaluated documents of *Priority* is much smaller than that of *BMW* when K is larger.

We now discuss how the number of evaluated documents in the *Priority* can be affected by K. The number of evaluated documents is determined by the sizes of all the M document blocks with the highest priority from the decision tree. The size of these blocks varies for different collections. For example, the size of the first block is the number of documents that containing all the query terms in the collection, i.e., all the documents that need to be evaluated with the conjunctive model. When K is larger than the size of the first block, the number of evaluated documents in *Priority* would often be smaller than those in *BMW* since the documents are prioritized using a simple priority function. However, when K is much smaller, the number of documents from the first block might be already much bigger than the value of K, which means that we need to unnecessarily evaluate many more documents than *BMW*.

When a query has more terms, the size of the blocks with higher priorities is smaller since these blocks require documents with more terms. As a result, these smaller blocks may lead to fewer documents that need to be evaluated by the *Priority* method. Table 6 shows the number of evaluated documents for different query lengths, the results are consistent with what we observed in Table 4. Clearly, the results are consistent with our analysis, which provides a justification on why our method is more efficient for longer queries.

4.5.3 Impact of the Tree-based Implementation

We have described a tree-based implementation for the proposed prioritization method as in Section 3.3. An alternative way of implementing the prioritization method is to simply allocate a block for each combination of query terms, sort the blocks based on the sum of IDF values of the corresponding query terms, assign documents to the block based on their priority scores through a binary search, and then apply the pruning strategy as described in Section 3.3. We refer this implementation strategy as a *non-tree* based implementation. One main advantage of the tree-based implementation is its ability to consider the relations among

Table 6: Avg num. of evaluated docs on *TB06L* for different query length (K=1,000)

Query length	2	3	4	5	6	7	> 7
BMW	**111,947**	109,859	93,399	107,984	113,466	96,659	100,398
Priority	229,752	**73,535**	**25,242**	**13,967**	**5,270**	**5,822**	**3,547**

Table 7: Avg. processing time per query (ms) for different query length on *TB06L* (K=1,000)

Query length	Avg.	2	3	4	5	6	> 6
Priority (tree)	101.9	86.3	88.8	94.7	126.6	152.3	220.8
Priority (non-tree)	179.5	66.9	83.1	85.5	124.7	150.8	3194.8

Table 8: Avg. query processing time (ms) on *TB06L*

K	100	1,000	5,000	10,000
Priority	**85.61**	**100.40**	**129.34**	**153.36**
Priority + No pruning	275.69	284.90	301.54	318.97
Exhaustive	1135.1	1141	1143.3	1154.4

blocks. For example, if a block is pruned, we do not need to access the blocks that are descendants of this block. On the contrary, it is difficult to capture the dynamic relations among the blocks in the *non-tree* based implementation.

Table 7 compares the performance of the two implementation methods on the *TB06L* collection when $K = 1,000$. Similar observations can be made for other collections and K values. We can see that the *tree-based* implementation has similar performance with the *non-tree* based implementation when the query is short, but it has a significant advantage when the query is longer.

4.5.4 Impact of the Pruning Strategy

Finally, we conduct experiments to examine how much the proposed pruning method in Section 3.3 can improve the efficiency. The results are shown in Table 8. The *Exhaustive* method refers to the query processing method that exhaustively computes the relevance score for every document containing at least one query term. It is clear that the prioritization method can improve the query processing significantly, and the proposed pruning method is able to further reduce the query processing time by a factor of at least 2. We also study how the percentage of pruned blocks changes as the query processing continues. The results are shown in Figure 9, where y-axis is the percentage of non-pruned blocks and x-axis is the percentage of the posting lists that have been processed. We can see that the percentage of non-pruned blocks keeps decreasing in the whole process, and around eight percent blocks are pruned at the end.

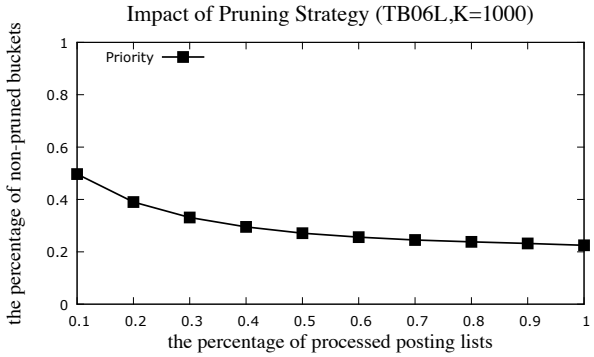


Figure 9: Impact of the pruning strategy: the percentage of non-pruned blocks

5. RELATED WORK

A large amount of research has focused on improving the efficiency of IR systems. We have provided background information about the query processing techniques and discussed the related studies in Section 2. We now provide information on other related work.

The basic idea of the proposed method is essentially cascade ranking [3,34,35], since we first apply a simple retrieval strategy (i.e., the priority function) to generate candidates and then use more complicated method (i.e., the given retrieval function such as Okapi BM25) to re-rank the candidates. The idea of cascade ranking has been recently studied in the context of learning to rank [3,34] and combining multiple retrieval signals [35]. Unlike these studies, we focus on exploiting this idea for scalable top-K query processing.

Tree-based data structure has been proposed as alternative index structure in recent studies [15, 20]. Culpepper et al. [15] proposed a hybrid approach based on FM-index, wavelet tree and the compressed suffix tree data structures for efficient query processing. Konow et al. [20] proposed to construct index based on the treap data structure. Instead of using the tree structure as a new representation for indexes, we focus on using tree-structure for more efficient query processing based on the inverted indexes.

6. CONCLUSIONS AND FUTURE WORK

Effectiveness and efficiency are two pillar requirements of large-scale IR systems. They both affect the satisfaction of search users. One of the major problems in the IR systems is to efficiently and effectively retrieve top-K ranked results. Two commonly used query processing methods include conjunctive and disjunctive mode. On the one hand, conjunctive is very efficient but with relatively lower effectiveness. On the other hand, the disjunctive mode can generate higher quality search results but is much slower.

In this paper, we propose a novel tree-based document prioritization method for query processing. The basic idea is to first prioritize documents using a simple yet efficient method and then pick only a small set of documents with higher priority for full evaluation. We propose to implement this idea using a tree-based structure. Experimental results show that the proposed method has a constant factor speed improvement over one of the state of the art dynamic pruning method for disjunctive mode. In particular, when the number of retrieved documents is larger than 1,000 or when the number of query terms is larger than 2, the proposed method can achieve a speed up around 2 over TREC Web

collections with marginal loss in terms of the effectiveness. We also find that it can improve the retrieval efficiency for Twitter collection with almost no effectiveness loss. Overall, the proposed method has been shown to achieve a better trade-off between the efficiency and effectiveness.

There could be many interesting directions for our future work. First, we plan to explore whether using other retrieval signals such as TF for prioritization can also lead to more efficient query processing methods. Second, we plan to conduct more experiments to examine what causes the performance difference between the two search domains. Third, we plan to extend our method to more complex query processing such as the ones related to proximity. Since proximity is often more computational expensive, our method might be able to achieve more speed up. Fourth, it would be interesting to study the performance modeling of the proposed method so that we could predict the query processing time for better online query scheduling. Finally, we plan to continue our efforts and study how to adapt the proposed method to the distributed environment.

7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Number IIS-1423002. We thank the anonymous reviewers for their useful comments.

8. REFERENCES

[1] V. N. Anh and A. Moffat. Compressed inverted files with reduced decoding overheads. In *Proceedings of SIGIR'98*, 1998.

[2] N. Asadi and J. Lin. Fast candidate generation for two-phase document ranking: Postings list intersection with bloom filters. In *Proceedings of CIKM'12*, 2012.

[3] N. Asadi and J. Lin. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. In *Proceedings of SIGIR'13*, 2013.

[4] R. Baeza-yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silverstri. The impact of caching on search engines. In *Proceedings of SIGIR'07*, 2007.

[5] L. A. Barroso, J. Dean, and U. Holzle. Web search for a planet: the google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[6] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of CIKM'03*, 2003.

[7] J. Brutlag, H. Hutchinson, and M. Stone. User preference and search engine latency. In *Proceedings of ASA Joint Statistical Meetings*, 2008.

[8] C. Buckley and A. Lewit. Optimizations of inverted vector searches. In *Proceedings of SIGIR'85*, 1985.

[9] S. Buttcher and C. L. A. Clarke. The trec 2006 terabyte track. In *Proceedings of TREC'06*, 2006.

[10] S. Buttcher and C. L. A. Clarke. Index compression is good, especially for random access. In *Proceedings of CIKM'07*, 2007.

[11] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. In *Proceedings of WSDM'10*, 2010.

[12] K. Chakrabarti, S. Chaudhuri, and V. Ganti. Interval-based pruning for top-k processing over comparessed lists. In *Proceedings of ICDE'11*, 2011.

[13] O. Chapelle, Y. Chang, and T.-Y. Liu. Future directions in learning to rank. *Journal of Machine Learning Research - Proceedings Track*, 14:1–24, 2011.

[14] N. Craswell, D. Fetterly, M. Najork, S. Robertson, and E. Yilmaz. Microsoft research at trec 2009: Web and relevance feedback tracks. In *Proceedings of TREC'09*, 2009.

[15] J. S. Culpepper, M. Petri, and F. Scholer. Efficient in-memory top-k document retrieval. In *Proceedings of SIGIR'12*, 2012.

[16] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. A candidate filtering mechanism for fast top-k query processing on modern cpus. In *Proceedings of SIGIR'13*, 2013.

[17] C. Dimopoulos, S. Nepomnyachiy, and T. Suel. Optimizing top-k document retrieval strategies for block-max indexes. In *Proceedings of WSDM'13*, 2013.

[18] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proceedings of SIGIR'11*, 2011.

[19] H. Fang, T. Tao, and C. Zhai. A formal study of information retrieval heuristics. In *Proceedings of SIGIR'04*, 2004.

[20] R. Konow, G. Navarro, and C. L. A. Clarke. Faster and smaller inverted indices with treaps. In *Proceedings of SIGIR'13*, 2013.

[21] T.-Y. Liu. Learning to rank for informaiton retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[22] C. Macdonald, N. Tonellotto, and I. Ounis. Learning to predict response times for online query scheduling. In *Proceedings of SIGIR'12*, 2012.

[23] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, 1996.

[24] I. Ounis, C. Macdonald, J. Lin, and I. Soboroff. Overview of the trec 2011 microblog track. In *Proceedings of TREC'11*, 2011.

[25] T. Qin, T.-Y. Liu, J. Xu, and H. Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):347–374, 2010.

[26] S. E. Robertson, S. Walker, S. Jones, M. M.Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of TREC-3*, 1995.

[27] C. Rossi, E. S. de Moura, A. L. Carvalho, and A. S. da Silva. Fast document-at-a-time query processing using two-tier indexes. In *Proceedings of SIGIR'13*, 2013.

[28] E. Schurman and J. Brutlag. Performance related changes and their user impact. In *Velocity - Web performance and operations conference*, 2009.

[29] T. Strohman and B. W. Croft. Efficient document retrieval in main memory. In *Proceedings of SIGIR'07*, 2007.

[30] T. Strohman, H. Turtle, and B. W. Croft. Optimization strategies for complex queries. In *Proceedings of SIGIR'05*, 2005.

[31] D. Takuma and H. Yanagisawa. Faster upper bounding of intersection sizes. In *Proceedings of SIGIR'13*, 2013.

[32] S. Tatikonda, B. B. Cambazoglu, and F. P. Junqueira. Posting list intersection on multicore architectures. In *Proceedings of SIGIR'11*, 2011.

[33] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Information Processing & Management*, 31(1):831–850, 1995.

[34] L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *Proceedings of SIGIR'11*, 2011.

[35] H. Wu and H. Fang. Tie breaker: A novel way of combining retrieval signals. In *Proceedings of ICTIR'13*, 2013.

[36] H. Yan, S. Ding, and T. Suel. Compressing term positions in web indexes. In *Proceedings of SIGIR'09*, 2009.

[37] J. Yang and J. Leskovec. Temporal variation in online media. In *Proceedings of WSDM'11*, 2011.

[38] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. In *Proceedings of WWW'08*, 2008.

[39] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), 2006.