Efficient Term Proximity Search with Term-Pair Indexes

Hao Yan[1*], Shuming Shi[2], Fan Zhang[3*], Torsten Suel[1], Ji-Rong Wen[2]

[1] Polytechnic Institute of New York University
[2] Microsoft Research Asia
[3] Nankai University, China

hyan@cis.poly.edu, {shumings, jrwen}@microsoft.com, v-fazhan@microsoft.com, suel@poly.edu

ABSTRACT

There has been a large amount of research on early termination techniques in the area of web search and information retrieval. Such techniques return the top-*k* documents without scanning and evaluating the full inverted lists of the query terms. Thus, they can greatly improve the efficiency of query processing. However, most existing research on early termination does not consider the impact of term proximity, i.e., the distance between term occurrences in a document, which has recently been integrated into a number of retrieval models to improve effectiveness. Only a limited amount of work has focused on how to improve query processing efficiency in this case.

In this paper, we propose new early termination techniques for efficient query processing for the case where term proximity is integrated into the retrieval model. We propose new index structures based on a term-pair index, and study new document retrieval strategies on the resulting indexes. We perform a detailed experimental evaluation on our new techniques and compare them with the existing approaches. Our experimental results on large-scale data sets show that our techniques can significantly improve the efficiency of query processing.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process; H.3.4 [**Systems and Software**]: Performance evaluation (efficiency and effectiveness)

General Terms

Algorithms, Performance, Experimentation

Keywords

Top-*k*, Term proximity, document structure, term-pair index

1. INTRODUCTION

A lot of research in web search and information retrieval has studied how to improve the efficiency of document retrieval, using techniques such as massive parallelism, caching, inverted index compression and early termination. We focus on one important class of optimizations, early termination techniques (also called dynamic index pruning techniques), which are widely used in IR systems and large search engines [31].

To better understand early termination techniques, we first look at the most basic index structure, the inverted index [31, 37]. An inverted index consists of many inverted lists, each of which is a sequence of postings. Each posting contains a document ID (docID), plus additional information such as the term frequency in the document, the exact positions of the occurrences, and their context (e.g., in the title, in anchor text, or in URLs). Typically the postings in each inverted list are sorted by their docIDs to achieve good index compression [31]. To process a query, a search engine could traverse the complete inverted lists for all relevant terms, calculate relevance scores for all documents in these lists, and finally return the top-*k* (e.g., *k* = 10) documents having the highest scores. However, such exhaustive evaluation requires significant computing resources and may greatly increase query response time.

To overcome this problem, many early termination techniques have been proposed [1, 2, 5, 10, 13, 17, 21, 24, 30, 31, 34, 35, 36]. The common goal of these is to speed up query processing by avoiding the processing of all documents in the relevant lists, and instead evaluating only a small subset. This is usually done by employing alternative index organizations such that during a traversal of these structures, the most promising documents (those likely to have the highest scores) are evaluated first while other documents may be evaluated later only as needed. Once a certain amount of documents has been processed, it is often possible to terminate the query evaluation and return the top-k results, without even considering the less promising documents.

The features being used to evaluate the documents (i.e., calculate the document scores) play a crucial role in the efficiency of early termination techniques, since they determine the best organization and ordering of the index, and thus the point at which early termination can occur. Most existing research on early termination techniques treats a document as a bag of words and evaluates queries using the following two kinds of features: (a) term-dependent features, e.g., within-document frequencies [21], or term-based IR scores or impacts [2]; and (b) term-independent features, e.g., Pagerank or other static ranks or scores [4, 17, 34] that measure the overall quality, importance, or popularity of a document (based on analysis of links, content, query logs, or traffic data in a preprocessing step). We note that while term-dependent scores are query-related, they are here only based on each separate term instead of the whole query, and thus do not depend on the relative positions or distances between terms within a document.

[*] This work was done when Hao Yan and Fan Zhang were interns at Microsoft Research Asia.

CIKM'10, October 26–29, 2010, Toronto, Ontario, Canada.
Copyright 2010 ACM 978-1-4503-0099-5/10/10...$10.00.

However, overall scores may also depend on the distance between the query terms in the document, called term proximity (TP), such that terms occurring close to each other often result in a higher score. In fact, the real search engine [4] has integrated the term proximity into their ranking system (although the details are not provided in [4]). In addition, a lot of recent research [7, 8, 15, 26, 28] has shown that retrieval effectiveness can be greatly improved by integrating term proximity scores into the retrieval model. Unfortunately, there is much less research on how to improve query efficiency for such proximity-aware retrieval systems, with the exception of [24, 35, 36]. In the following, we will refer to early termination (ET) techniques that consider term proximity (TP) as TP-ET methods, and refer to those without TP as NTP-ET methods. We note that while real search engines often integrate into their overall scoring function a variety of other features beyond static ranks, term-based IR scores, and query-based term proximity scores, in this paper we only focus on these three kinds of scores, which we call SR, IR, and TP scores, respectively.

Thus, the study of TP-ET techniques is interesting and important due to the importance of term proximity factors in state-of-the-art ranking functions. However, the index structures and retrieval strategies of existing NTP-ET techniques can not be directly applied to TP-ET methods. The main reason is that each inverted list is only associated with one particular term and does not consider any other terms, while the TP score is based on the entire query and therefore depends on the interaction between several query terms. Independently ordering each relevant inverted list of a given query in some order, say by term scores, may result in a fairly non-monotonic and almost random distribution of the TP scores that makes early termination impossible for most queries. Thus, the main challenge for TP-ET methods is how to consider the impacts of all three kinds of scores to achieve effective early termination and thus efficient query processing.

In this paper, we study new early termination techniques that improve retrieval efficiency for the case where term proximity information is taken into account in the retrieval model. Our goal is to create a new auxiliary index structure and mechanism that can be used in IR systems to speed up query processing without reorganizing their entire structure. In particular, we create an additional term-pair index for cases where certain pairs of terms occur close to each other in a document and propose new retrieval strategies for the resulting indexes. The new index organization implicitly moves documents with high term proximity scores towards the front of the query processing pipeline, without disturbing the normal indexes too much. Thus, the documents with the highest overall scores are likely to be evaluated first during query processing, resulting in effective early termination. Our experimental results show that our methods can achieve significant improvements in efficiency over existing methods.

2. BACKGROUND AND RELATED WORK

We refer to [31, 37] for basic background on indexing and query processing in search engines.

2.1 The Ranking Functions

As mentioned in Section 1, we focus on the following three types of scores: SR, IR and TP scores. Almost none of the existing research on early termination techniques has studied other additional types of scores beyond these, though real search engines may do so. In fact, most ET techniques are based on only one or two of these. For example, [2, 21] uses only the IR score while [4, 17, 34] considers both SR and IR scores and [24] studies both IR and TP scores. There are only a few ET techniques [35, 36] that have integrated all three scores into their ranking functions. The overall document score for a particular ET method is often evaluated as a linear weighted sum of all types of scores considered by it, and the general ranking function for most of the ET techniques is as follows:

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

where $S(d, q)$ is the overall score of the document d with respect to the query q, $SR(d)$ is the SR score of the d, $IR(d, q)$ and $TP(d,q)$ are respectively the IR and TP scores of the document d with regard to the query q, while α, β, and γ are three non-negative parameters ($\alpha + \beta + \gamma = 1$). Usually all of the SR, IR and TP scores are normalized into the range [0, 1]. Formula (2.1) can be adapted in various ways by tuning α, β, and γ. For example, ranking functions for methods that only use SR and IR scores can be modeled by setting $\gamma = 0$.

There has been a lot of research on the calculation of each of the three types of scores. The SR score could be computed using the Pagerank method in [4] but could also incorporate various other measures of document quality or importance. One popular way to calculate the IR score is the BM25 formula in [23], which has been widely used in IR systems. However, the calculation of TP scores is often more complicated. It does not depend only on a particular term but on the entire query. Many approaches [7, 8, 15, 26, 28, 22, 24, 35, 36] have been proposed to calculate TP scores. Most methods assume the TP score of a pair of occurrences to be inversely proportional to the square of their distance within the document, but the concrete implementations are different from each other and the ways to combine such pair TP scores into the document TP score are also different. However, a popular way is to first slide a window with a certain size w over the document, and then each time calculate the TP score for a term pair $<q_t, q_k>$ based on only the contributions from the occurrences of q_t and q_k within that window. Then all such pair scores are combined using a weighted sum, to obtain the final document TP score.

The ranking functions of practical search engines also take into consideration the document structure and the context of term occurrences, e.g., whether they are in the title, or in the URL, for better result quality [4]. Like [4, 35], we distinguish the following four different contexts (we call them fields) of a web document: title, URL, anchor (text), and body fields, where the anchor text refers to the visible, clickable text (in other pages) in a hyperlink pointing to the page, while the body field refers to the rest of the web page (anything not in the other three fields).

2.2 Early Termination without TP

Ideally, an early termination technique stops evaluating documents immediately once the top-*k* documents have been discovered. In practice, we cannot immediately tell if a document we just encountered will be in the final top-*k*, and thus we have to continue evaluating new documents until we are sure that no new document can achieve a higher score than any document in the current top-*k* list. In addition, we often require that the *k* documents in the result list (achieved by the early termination techniques) are returned in the same order as without early termination.

We note that although many early termination methods may relax the above restriction allowing for approximate top-*k* results [5, 10, 17, 32] (e.g., the result list contains 99% of the real top-*k* documents on average) as long as a certain retrieval precision can be reached, we only focus on exact top-*k* query processing, that is, all top-*k* results must be returned and in the correct order.

Index Reorganization: Most early termination techniques reorganize the inverted lists in some way that is ordered by certain types of scores, such that the most promising documents are skewed towards the beginning of the lists, and thus evaluated earlier than other documents. In particular, the method in [21] does so based on the within-document frequencies (which are assumed to dominate the IR scores). The method in [4] stores the postings (hits) of a list into two sets of inverted barrels: one set for the hit lists that include title or anchor hits and another set for all hit lists. The method in [2] partitions an inverted list into *m* segments in each of which all documents are of the same impact values (which are essentially quantized IR scores) and sorted by docIDs. The segments themselves are sorted in descending order of their impacts. The approach in [17] partitions the documents in a list into two segments based on their IR scores, and the segment with the higher scores is evaluated first. All documents within each segment are sorted in descending order of their SR scores. In this way, the documents with the highest IR and SR scores are located either in the top segment or the beginning of the bottom segment. The very recent research in [34] sorts a list by a combination of the so-called UBIR score and the Pagerank (or static rank), which are both term-independent information.

Retrieval Strategies: Many evaluation strategies [2, 5, 14, 17, 19, 30, 32, 34, 35, 36, 37] have been proposed in the IR and web search areas, and they can be roughly divided into the following three categories: document-at-a-time (DAAT) [5, 14, 17, 30, 32, 35, 36, 37], term-at-a-time (TAAT) [19, 30, 37] and score-at-a-time (SAAT) [2]. DAAT evaluates a document by considering the contributions of all query terms, before it deals with the next document; TAAT evaluates all documents in the inverted list of one term before it does so for the next term; SAAT is only suitable for indexes sorted by impacts [2]. While TAAT is widely used in the traditional IR systems and SAAT can achieve good performance in certain cases [37], DAAT has been shown to be able to achieve very good query performance in many cases especially with certain optimizations [5, 14, 17, 30, 32, 35, 36, 37]. DAAT often requires a smaller run-time memory size while the other two methods need more memory to maintain intermediate scores during query processing. Please refer to [2, 5, 37] for a detailed comparison among those strategies.

We note that many retrieval algorithms have also been proposed in the database area, besides the IR and web search areas, e.g., Fagin's Algorithm (FA) [11], the imporved Threshold Algorithm (TA), and the No Random-access Algorithm (NRA) [12]. The main difference between TA and NRA is that the former allows random access on the inverted list, while the latter only allows sequential access (Please refer to [13] for a survey of these methods).

2.3 Early Termination with TP

There are only a few early termination approaches [24, 35, 36] (and [4] although the concept of early termination was not explicitly presented in it) that integrate the TP information into their retrieval models. They adopt different strategies to overcome the above problem, where the methods in [4, 35] exploit the document structure to reduce the upper bound of the unseen scores, while [24, 36] implicitly move the documents with high TP scores to the front of the list by creating new term pair indexes. In particular, the method in [4] groups the documents of a list into two sets where one set is actually a subset of the other one and contain only those hit lists that include title or anchor hits. That is, they assume that the occurrences in the title or anchor fields imply high IR scores and therefore should be evaluated first. The method in [35] also exploits the document structure information to organize the indexes. It partitions each list into the following two segments: one top segment containing the postings only for the occurrences within the three fields of title, anchor text, and URL, and another bottom segment containing the postings only for those within the body field. During query processing, it first processes the entire top segment, and then attempts to achieve early termination in the second segment, based on the fact that parts of the TP scores (associated with the title, anchor and URL fields) have been calculated in the top segment and thus the upper bounds of the TP scores for all unseen documents in the second segment can be reduced.

In contrast, [24] and [36] approach the problem from another angle: They create additional indexes for pairs of terms in the document and exploit those indexes to implicitly move documents with higher TP scores to the front of the lists. In particular, [24] creates additional indexes for all possible term pairs, i.e., pairs with any possible distances between each other within the same document, while [36] only creates such indexes for the phrases. However, [24] sorts indexes only by TP scores, and does not consider the impact on index organization due to SR and IR scores (although its CL method evaluates the document by both IR and TP scores). In addition, the size of its additional pair indexes is huge as it builds indexes for all combinations of terms of a document (while [24] mentions a static pruning method to reduce this index size, the details of the method are not provided). Moreover, [24] does not consider the SR score and is thus not applicable to cases such as the web where overall document quality and importance is often as important as term-based scores. Finally, it is interesting that [36] has shown that we can improve efficiency greatly by creating additional extended hit-list only for some selected phrases in the document, without exploding the index size. However, [36] does not study the potential benefits of using term pairs with larger distances, which may improve the query efficiency a lot more although it may complicate both index construction and document retrieval (the details will be discussed in later sections). Moreover, neither of [24] and [36] consider the document structure, which as discussed above has been used in the real search engines [4]. In addition, both of them have to do full evaluation of documents that is desirable to be avoided.

2.4 Other Related Work

Compared to the above dynamic pruning techniques, static pruning techniques (e.g., [6, 20]) try to predict and discard certain less important parts of the index structures as the indexes are being built. Such methods achieve high retrieval efficiency by sacrificing on search quality for some queries. The method in [3] creates the auxiliary indexes for firstword-nextword pairs to speed up the phrase query. However, it is not directly suitable to the non-phrase query. The pre-aggregation techniques [16] first pre-aggregate the intersections of the lists and then simultaneously process the intersection list and the term lists to speed up the retrieval. Interestingly, [18] also uses the intersection lists as an intermediate level of a three-level caching structure to speed up query processing. However, the intersection lists in [16, 18] do not contain the position information of terms. Some other early termination techniques [5, 32] focus on reducing the number of full evaluations. Their main idea is to first evaluate all documents using approximate scores and then perform the full evaluation only on the documents with the highest approximate scores. However, we often calculate all of the SR, IR and TP scores unless we can safely avoid doing so without loss of accuracy. Finally, early termination strategies are also affected by caching policies [18, 25, 29]. In this paper, we only focus on dynamic pruning techniques to get the exact top-k query results and do not consider pre-aggregation and caching policies.

3. CONTRIBUTIONS OF THIS PAPER

In this paper, we study and evaluate efficient document retrieval techniques for the case where term proximity information is integrated into the retrieval models. Our goal is to provide the search engine with a separate component to speed up the query processing greatly while not incurring much overhead of storing the extra indexes. Our main contributions are as follows:

(1) We propose new index structures by creating additional term pair indexes for pairs of terms that are within certain distances to each other in the documents, and study corresponding retrieval strategies for the resulting indexes. We also proposed new methods to reduce the index size of the term pair indexes.

(2) We integrate the impacts of document structure information, i.e., the context of term occurrences, into our retrieval models. Although most of the existing research on early termination techniques does not consider such information, the real search engine [4] does so. In particular, we take into consideration the following four fields of the document structure: title, anchor, URL and body fields.

(3) We propose the new methods to avoid full evaluations on the TP scores by using our term pair indexes. Thus our methods can not only reduce the number of documents to be evaluated during query processing, but also save the computation cost by avoiding unnecessary full evaluations.

(4) We compare our algorithms with other existing techniques on large scale data sets. Our experimental results show that our approach can consistently improve the query efficiency significantly while also achieving a reasonable tradeoff between query efficiency and index size.

4. OUR ALGORITHMS

Our goal is to improve the query efficiency especially on the proximity-aware retrieval models by creating for the search engines an auxiliary index component (term pair indexes) which can be easily plugged in the existing systems. Therefore, our new index architecture is composed of the normal indexes, which may be organized by any methods discussed in Section 2, and the term pair indexes. We note that the new pair indexes do not change the index organization of the normal inverted indexes.

The main idea of our algorithms is: we exploit the additional term pair indexes to implicitly move the documents with the highest TP scores on top of other documents in the normal indexes. Recall that the normal indexes are not affected by the pair indexes and often have been organized by other early termination techniques discussed in Section 2, such that the documents with the highest SR or IR scores are located to the beginning of the normal indexes. Therefore, under our new architecture, the most promising documents (with the highest integrated scores of SR, IR and TP scores) are organized as the first tier of documents to be evaluated and thus the early termination can be expected.

A query under the new architecture is then processed as follows: when the engines receive a query, they first load and process the relevant lists from the pair indexes (as long as they contain such relevant lists); they then load the normal inverted indexes and continue to evaluate the documents of these lists until the top-k results can be safely returned without scanning the entire lists.

4.1 The Ranking Function

Our ranking function is based on the formula (2.1) discussed in Section 2. However, we also integrate into it the document structure information for the following four fields of a web page: title (T), URL (U), anchor (A) and body fields (B). In particular, we represent the IR score (or the TP score) as the weighted sum of its partial scores in all of the four fields (we note that unlike IR and TP scores, the SR score is not affected by the document structure). Therefore, our ranking function can be described as follows:

$$
\begin{aligned}
S(d,q) = \alpha \times SR(d) + \beta \times \sum_{i\epsilon(T,U,A,B)} \left(\omega_i \times IR(d,q,i)\right) \\
+ \gamma \times \sum_{i\epsilon(T,U,A,B)} \left(\omega_i \times TP(d,q,i)\right)
\end{aligned}
\tag{4.1}
$$

where ω_i is the weight for the ith field, the $IR(d,q,i)$ and the $TP(d,q,i)$ are respectively the partial IR and TP scores of the query q in the ith field, while other symbols are of the same meanings as those in the formula (2.1).

We now discuss how to calculate various scores. The SR scores can be achieved in the exact same way as in [35, 36]. The IR partial scores can be calculated by the BM25 formula [23] except that they are computed based on the term occurrences in a particular field instead of those in the entire document. The basic process of calculating the document TP score has been discussed in Section 2 and is based on all pair-wise occurrences of query terms within a fixed-size window. The scoring function for a particular pair-wise occurrence can be derived from the scoring models in [35, 36, 24, 7, 8] and one of their common features is that such a score is inversely proportional to the square of the distance between terms as follows:

$$
TP(t_1,t_2) = f\left(\frac{1}{1+dist(t_1,t_2)^2}\right) \tag{4.2}
$$

where $TP(t_1,t_2)$ is the TP score for one particular pair-wise occurrence of the terms $t_1$and t_2 , while $f()$ is a linear function of the square of the distance $dist(t_1,t_2)$ of the two terms. The value of $TP(t_1,t_2)$ is also affected by the ordering of the occurrences in the document and that in the query. For example, given a query "New York", we will assign a higher $TP(t_1,t_2)$ score to the occurrence of "New York" than that of "York New". This can be achieved by representing the $dist(t_1,t_2)$ as followings

$$
Dist(t_1,t_2) = \left|(p_{t_2}^d - p_{t_1}^d) - (p_{t_2}^q - p_{t_1}^q)\right| \tag{4.3}
$$

where $p_{t_1}^d$and $p_{t_2}^d$are the positions of t_1 and t_2 in the document while $p_{t_1}^q$and $p_{t_2}^q$are their positions in the query.

4.2 Building New Indexes

In this subsection, we first describe how we build the term pair indexes and then discuss how they are combined with various index structures of the normal indexes.

Building term pair indexes: Given all of the relevant documents to a query, our goal is to create additional indexes for a small subset of them that contain the close-by term pairs and thus potentially have the highest TP scores. In particular, given a term pair (t_1, t_2) (which is different from another pair (t_2, t_1)), we first identify the documents which contain at least one pair-wise occurrence of them with $dist(t_1, t_2) \leq m$, where $dist(t_1, t_2)$ is derived from the above formula (2.4) and m is a certain distance value, say $m = 3$. The possible forms of such occurrences are: $t_1 t_2\ (dist = 0)$, $t_1 t_x t_2 (dist = 1)$, $t_1 t_x t_y t_2 (dist = 2)$, and $t_1 t_x t_y t_z t_2 (dist = 3)$, where t_x, t_y, t_z can be any other terms in the document but t_1 and t_2. We then build an inverted list (we call it term pair list) for the pair (t_1, t_2) based only on the above identified documents. The basic form of such a term pair list with n postings is shown in Figure 4-2.

$$
\begin{aligned}
\{\ \ & < did^1, f_{tp}^1, \left[(T_1^1, P_1^1), (T_2^1, P_2^1), \ldots, \left(T_{f_{tp}}^1, P_{f_{tp}}^1\right)\right] >, \\
& < did^2, f_{tp}^2, \left[(T_1^2, P_1^2), (T_2^2, P_2^2), \ldots, \left(T_{f_{tp}}^2, P_{f_{tp}}^2\right)\right] >, \\
& \ldots., \\
& < did^n, f_{tp}^n, \left[(T_1^n, P_1^n), (T_2^n, P_2^n), \ldots, \left(T_{f_{tn}}^n, P_{f_{tn}}^n\right)\right] >\ \}
\end{aligned}
$$

Figure 4-2 The basic form for a term pair list of *n* postings

The term pair list is then a sequence of postings, each of which contains a docID, the term pair frequency f_{tp}, and all of the f_{tp} occurrences. Unlike the posting in the standard inverted list discussed in Section 1, which always records the occurrences of a single term in a document, the posting in the term pair list does so for either one or both of terms in the query, according to the distances between them. In particular, we treat two consecutive occurrences of them with $dist(t_1, t_2) \leq m$ as a single occurrence of the pair, while we encode those with larger distances as two separate occurrences. Thus each occurrence can be represented as a code of (T, P), where T stands for one of the following types of the occurrences: (1) t_1 and t_2 appear together (and in the order of (t_1, t_2) rather than (t_2, t_1)) with $dist(t_1, t_2) = c$ and $c \leq m$, (2) t_1 appears by itself (i.e., its $dist$ to the closest following t_2 is greater than m), (3) t_2 appears by itself (i.e., its $dist$ to the closest previous t_1 is greater than m); while P stands for the position of the first term in case (1), or the position of the term in the other two cases. Thus the value of f_{tp} is the number of all of such occurrences. Please note that we build only a single list for the pair (t_1, t_2), while we treat (t_1, t_2) and (t_2, t_1) as different pairs and will create separate term pair lists for them.

From the above we can see that on one hand, we never maintain any information in the term pair list for the documents that contain no pairs of (t_1, t_2) with $dist(t_1, t_2) \leq m$, while on the other hand, once a document d contains such a close-by pair, the information for all occurrences of both terms within d will be encoded into the list. The reason we keep all such information is to provide the flexibility for the search engines to employ various ranking functions and evaluation strategies on the term pair indexes. Once the engines fix such settings, a non-trivial amount of redundant information in the term pair indexes can be safely removed without downgrading the query performance much (details will be discussed soon).

Cooperation with normal indexes: Before being combined with the above term pair indexes, the normal inverted indexes often have been reorganized by a variety of other early termination methods discussed in Section 2 (especially in Subsection 2.2). In this paper, we mainly focus on the following three kinds of index organizations of the normal inverted indexes: (1) the standard inverted indexes structure [31] (which we call STD indexes) where postings are sorted by docIDs or SR scores (as discussed in Section 2, the latter ordering can be easily transformed into the former one with docID reassignment in the order of the SR scores); (2) the index structure in [17] (which we call HL indexes) where both SR and IR scores are considered, resulting in the two segments with high and low IR scores respectively, in each of which all postings are sorted by SR scores; (3) the index structure in [35] (which we call structured or STR indexes) where all of SR, IR and TP scores are considered and the indexes are also divided into two segments (the TAU segment and the B segment) but according to whether the occurrence happens in the TAU (title, anchor and URL) fields or in the B (body) field. We note that although there are some other index structures (e.g., sorting postings only by the TP scores [24]) that may have been used for the normal inverted indexes, the above three ones, i.e., STD, HL, and STR, can to some extent represent most of the index organizations used in the state-of-the-art early termination techniques with or without consideration of the TP information. For example, as discussed in Section 2, the HL structure can be easily converted to a special case of the structure in [2], while both STD and HL are considered for the methods in [36].

4.3 Retrieval Strategies

In this subsection, we discuss the retrieval process for our new index architecture (using the two-term query as an example), where the normal indexes are organized in either one of STD, HL and STR index structures.

Given a query q of (t_1, t_2), our retrieval process is divided into the following two phases:

(1) In the first phase, we check the term pair indexes to see if they contain the list $l_{t_1,t_2}$for (t_1, t_2) . If they do not do so, we skip the rest part of this phase and go directly to the second phrase; otherwise we load l_{t_1,t_2}into the memory. We then load the list l_{t_2,t_1} if it also exists in the term pair indexes. After that, we process the entire l_{t_1,t_2} and l_{t_2,t_1} evaluating appropriate documents in them, resulting in a temporary top-k list in which all documents are of their complete scores and do not need to be evaluated again in the second phase.

(2) In the second phase, we load the normal indexes and continue to evaluate new documents in them (skipping the documents that have been evaluated in the first phase) in a DAAT manner until the early termination can be achieved.

From the above, we can see that all documents with the high TP scores have been completely evaluated in the first phase and put in the temporary top-*k* list. As a result, the upper bound of the TP scores for all documents in the second phrase is greatly reduced since none of them contain close-by pairs. Thus as long as the index structure of the normal indexes (in the second phase) has the property that the documents with the high SR and IR scores are also located at the beginning of the lists, the quick early termination in the second phase can be expected in such cases since the early termination condition discussed in Subsection 2.2 can be quickly satisfied after a small amount of documents have been evaluated.

Interestingly, the term pair indexes can not only be used to reduce the number of documents to be evaluated (since only a small proportion of the lists need to processed before early termination), but also be exploited to save the number of full evaluations on TP scores as follows: when a new document is encountered, we often have known its exact SR and IR scores (that can often be pre-computed since their values do not depend on other terms in the query) and the upper bound of its TP score, therefore we can easily get the upper bound of its overall document score, which can then be compared with the score of the kth document in the current top-k list. Once we find its overall score is smaller than that of the kth document, we can safely discard it and thus avoid the expensive full evaluation of its TP score.

For the queries with more terms, we present a very simple method by taking advantage of the nonexistence of term pair lists as follows: if there are no existing term-pair lists for any pair of the query $(t_1, t_2, t_3, \dots, t_n)$ (and this is not due to the term-based pruning of the lists), we then know that there are no documents with very high TP scores and therefore the upper bound of the TP score for all unseen documents can be reduced and thus the early termination may be achieved.

4.4 Index Size Reduction

The term pair indexes can be pruned in either a term-based manner and in a posting-based manner. First, we do not need to build the pair lists for all term pairs based on the rareness of the terms and the pairs. For example, if the normal inverted lists for both terms are very short (which means both of them are rare terms), we do not need to build the pair list for them since it will not take much time to process even the whole normal lists of them. In contrast, it is always desirable to build the pair list for a rare pair that is composed of two common terms since the pair list will then be much shorter than either of the term lists.

Alternatively, another interesting way to reduce the size for the term pair indexes is to prune the number of postings stored in each term pair list, without affecting the precision of the top-k results. As discussed above, if the ranking functions (and its parameters) of the retrieval models are fixed, we do not need to store a lot of information in the current term pair lists, while we can still achieve the same results in the first phase of our current retrieval models. This can be achieved by the following: we pre-compute the top-k list for all documents of the term pair lists (those processed in the first phase) during the index construction period and thus we only need to store the resulting top-k list for those documents in the term pair list, along with a hash table specifying which documents in the lists have been processed and thus will not be reevaluated in the second phase. In our experiments, this idea is slightly modified since we want to keep the ranking functions as flexible as possible. In particular, for each posting in a term pair list, we only keep the position information for close-by pairs while we discard the position information of the independent occurrences of the single terms (we do this only for the body field and still keep all position information in the other three fields since the body field dominates the size). Our later experiments will show that the index size can be greatly reduced by using the above various optimizations.

In addition, on one hand, we can reduce the size of the resulting pair indexes even further using a better compression approach (e.g., PForDelta in [33]) to compress docIDs and frequencies; while on the other hand, we can also improve the compression performance for positions, based on the observation [32] that the clustering property existing in the single-term occurrences can lead to better compression for positions. In fact, we may expect to achieve even better compression ratio since the correlation between consecutive pairs may be stronger than that between successive single terms.

5. EXPERIMENTS

5.1 Experimental Setup

For our experiments, we use the following three data sets: the widely used TREC GOV (1.25 million web pages), TREC GOV2 (25.2 million web pages), and a newly distributed TREC ClueWeb09 data sets [9] which consists of 1.04 billion web pages in ten languages while only those in English, about 500 million pages, are used in our experiments. For the evaluation on the GOV data set, we use the *trec2004mixed* query set which contains 225 queries and 51 two-term queries among them; for the ClueWeb09 data set, we use the query track (we call *trec2009mq*) of TREC2009 which contains 40,000 queries and 14,620 two-term queries among them. For the GOV data set, we use a single machine with Dual 2.13 GHz Intel Core^{TM}2 CPU, 4GB RAM, and 2*500 GB local SATA disk. For the ClueWeb09 data set, we use 40 machines, where each machine has Quad 2.50 GHz Intel Xeon CPU, 16GB RAM, and 1.5 TB or 4TB local disks. All web pages are distributed to those machines via URL hashing.

5.2 Experimental Results

We first compare in Table 5-1 the least number of documents (in percentage of the list size) to be evaluated (i.e., we assume that we magically know where the top-k documents are in the inverted lists) on the GOV data set using our new index architecture where the normal indexes are organized as STD, HL or STR indexes. Since the locations of all top-k documents are magically known, the query processing can be immediately terminated once all of the top-k documents have been scanned. Therefore, the results show the potential that the best early termination techniques can achieve under our architecture with different term distances (i.e., the value of m in Subsection 4.2) and various normal index structures.

Table 5-1. Average percentage (%) of evaluated documents using the magic early termination for the GOV data set

Index Structure	W/O Term Pair Indexes	W/ Term Pair Indexes		
		m= 1	m= 2	m = 3
STD	47.46	4.26	1.17	1.0
HL	38.06	3.00	1.20	0.99
STR	12.48	1.74	1.41	1.36

From Table 5-1, we can see that our methods with the term pair indexes can significantly reduce the number of documents required to be evaluated by other early termination methods without them, for all of the three kinds of normal indexes. This implies that using term pair indexes can potentially achieve much faster early termination and thus much more efficient query processing. For example, for the STD indexes, our methods with term distance m=3 only need to evaluate 1% of all documents in the lists, while those methods without term pair indexes need to process half of the entire lists.

More interestingly, in our methods, using a larger term distance (for the term pair indexes), e.g., m=3, can result in much less number of evaluations than using a smaller distance, e.g. m=1. The reason is that (as discussed in Section 4) once the term pair indexes are fully processed, the upper bound of the TP scores for all unseen documents to be evaluated in the normal indexes can be reduced much more in the former case than in the latter case. This observation motives us to exploit the term pairs with larger distances to improve the query performance (as long as the extra index size is acceptable). Based on the above observation, we expect to achieve similar query performance in the following experiments using our real early termination techniques, where query processing cannot be stopped until either the early termination condition (discussed in Subsection 2.2) is satisfied, or the entire lists have been completely processed.

Table 5-2. Query processing time (ms/query) on the TREC GOV data set

Index Structure	W/O Term Pair Indexes	W/ Term Pair Indexes		
		m= 1	m= 2	m = 3
STD	158	132	62	56
HL	132	61	46	34
STR	50	32	30	32

In Table 5-2, we compare the query processing time (ms/query) for our methods with term pair indexes and those methods without them on the GOV data set, where all methods use the real early termination. From Table 5-2, we can achieve the following observations: First, as expected, our methods can achieve much faster document retrieval than the methods without the term pair indexes and our methods using the term distance m=3 can result in the best performance with only 32 ms / query on the STR normal indexes. Second, as we have shown in the results for the magic early termination, using the farther term distance (m=3) can achieve faster query processing than using the distance of m=2 than that of m=1.

More interestingly, our methods can greatly narrow the performance gap between different normal index structures. For example, the difference of the retrieval speed among the methods using the STD, HL and STR normal indexes are largely reduced by using our method with the term distance m=3. This observation shows that our method may in general be used as a flexible and helpful component for the search engines to improve the query efficiency without worrying much about how the normal indexes are organized themselves.

Another interesting observation for Table 5-2 is: although using the term distance m=3 can result in significant improvement over using m=1, it can only achieve slightly better performance than using m=2. This implies that it might not be much beneficial to build the term pair indexes with a very large term distance since in that case the gain of the faster processing speed may be outweighed by the overhead of the extra index size (the tradeoff will be discussed in more details soon). We perform similar experiments on the TREC GOV2 and ClueWeb09 data sets, and similar results can be achieved and are not displayed due to the space limitation.

Table 5-3. The number of evaluated documents during query processing on the GOV data set

Document Numbers		W/O Term Pair Indexes	W/ Term Pair Indexes		
			m = 1	m = 2	m = 3
n_{all}	**STD**	4,452,906	3,653,919	1,555,227	1,306,699
	HL	3,217,435	1,128,684	583,827	211,216
	STR	770,742	477,671	459,844	461,674
n_{aux}	**STD**	0	109,902	114,919	123,185
	HL	0	109,902	114,919	123,185
	STR	0	109,902	114,919	123,185
n_{TP}	**STD**	5,281	3,890	3,917	4,008
	HL	4,971	3,865	3,956	3,630
	STR	3,242	400	137	93

In Table 5-3, we show the total number of documents (n_{all}) (associated with the 51 two-term queries in the trec2004mixed query set) that are evaluated on the GOV data set during query processing, for all methods compared in the previous tables. We also show the number of the documents (n_{aux}) that are evaluated in the auxiliary term pair lists and the number of documents (n_{TP}) whose TP scores are fully evaluated. From Table 5-5, we can see that although our methods need to first process the additional term pair indexes, we evaluate much less number of documents in the normal indexes than the methods without the term pair indexes, which is the main reason that we can achieve faster query processing speed than them. Interestingly, we can also see that although using the larger term distance may lead to evaluating more documents in the term pair indexes than using the smaller distance, the total number of documents evaluated by them is much smaller. Therefore, our method using the larger term distance can achieve the faster speed of query processing than that using the smaller term distance. In addition, we observe from Table 5-3 that using our term pair indexes can also help to save the number of full evaluations on TP scores (i.e., the value of n_{TP}) due to the reasons discussed in Section 4. The similar experimental results can also be achieved from the ClueWeb09 data sets and are not shown here due to the space limitation.

We now show the experimental results for the impacts on the query efficiency of using various parameter values in our ranking functions. Recall that in the basic form of our ranking function (formula (2.1)), there are three parameters α, β, and γ, specifying the weights of the SR, IR and TP scores respectively. The higher weight for a certain kind of score implies the kind of scores may have a greater impact on the overall document score than other kinds of scores. We note that the SR score is independent of the terms, while both the IR and TP scores depend on the terms and are often correlated with each other (we also find such correlation between them through our experiments). Therefore, we slightly changed the formula (2.1) such that $\gamma = k \times \beta$, where k is parameter achieved from our experiments and still $\alpha + \beta + \gamma = 1$. We are going to show the experimental results in terms of either various values of α or various values of the rate γ/β . First, we compare in Table 5-4 the query processing time (ms/query) on the ClueWeb09 data set (with the STR indexes), using our methods and the methods without the term pair indexes (w/o TPI), in terms of different values of γ/β and a fixed value of $\alpha = 0.2$.

From Table 5-4, we can achieve the following observations: First, our methods with term pair indexes can consistently achieve much faster query processing than the methods without them for various values for the weights of TP scores (we can also achieve such an observation in Table 5-5 that will be explained soon). Second, our methods using a larger distance can achieve better performance with the increasing of the weights of TP scores. Similar observations can also be achieved from the experimental results on the GOV data set and are not shown here due to space limitation.

Table 5-4. Query processing time on ClueWeb09 (STR indexes), for the various values of γ/β and a fixed $\alpha = 0.2$

γ/β	W/O Term Pair Indexes	W/ Term Pair Indexes		
		m= 1	m= 2	m = 3
0	145	137.	129	125
0.25	147	118	109	106
0.5	147	116	105	101
1	147	109	100	92
2	144	100	87	85
4	141	94	85	80

Table 5-5 . Query processing time on the ClueWeb09 data set (STR indexes), for the various values of ω_{TAU}/ω_B

ω_{TAU}/ω_B	W/O Term Pair Indexes	W/ Term Pair Indexes		
		m= 1	m= 2	m = 3
0.25	154	100	90	85
0.5	155	108	98	91
1	147	109	100	92
2	148	112	102	98
4	144	110	105	99

We also compare in Table 5-5 the query processing time on the ClueWeb09 data set (with the STR indexes) in terms of different rates of the weight for the TAU (title, anchor and URL) fields and that for the body field (i.e. ω_{TAU}/ω_B). From Table 5-5, we can see that when the TAU fields dominate the overall scores, we can achieve faster early termination since the indexes for the TAU fields are always evaluated earlier than those for the body field.

Table 5-6. The index size of the term pair indexes that are only associated with the *trec2009mq* query set on the ClueWeb09 data set, and the total Index size per machine

Index Type		Index-size for trec2009mq			Total Index-size Per machine		
		m=1	m<=2	m<=3	m=1	m<=2	m<=3
Normal Indexes (STD, HL, or STR)		18 GB			60 GB		
Term-Pair Index	W/O Optimization	2.27 GB	3.25 GB	3.93 GB	>0.5 TB	>1.0 TB	>1.5 TB
	With Optimization	410 MB	503 MB	573 MB	41.8 GB	75.1 GB	104 GB

Finally, we compare the index size of normal indexes and our term pair indexes for the ClueWeb09 data set in Table 5-6. We first show in the middle column the index size only for the queries in the *trec2009mq* query set. This is a rough measure of the amount of additional data per query that has to be transferred from the disk to the memory [32] when employing term-pair indexes. We then show the index size per machine for the whole ClueWeb09 data set. We show the index size for the term pair indexes with and without the optimization (the index reduction techniques) discussed in Subsection 4.4. From the table, we can see that, the size of the term pair indexes is greatly reduced using our methods. We note that the index sizes in both the above tables can be further reduced by applying better compression methods [32,33]. In addition, if we also take into consideration of the caching techniques, large proportion of disk traffic can be avoided and therefore the overall good query efficiency can still be achieved.

6. CONCLUSION AND FUTURE WORK

In this paper, we have studied early termination techniques for the proximity-aware retrieval models on large-scale data sets. We proposed a new index which essentially offers the current search engines an additional component (term pair indexes) that can improve the query efficiency greatly without changing the original inverted indexes. Our experimental results show that our methods can significantly improve query efficiency especially for the proximity-aware retrieval models.

There are still several interesting open problems. First, besides the simple method we mentioned in Subsection 4.3 to deal with multiple-term queries, we are currently studying other methods for such queries. There are two intuitive methods that might be used to extend our methods for the multiple-term queries: one is to directly build the additional multiple-term indexes instead of the pair indexes, while the other is to first decompose the multiple-term query into a set of two-term queries and then combine the results of those two-term queries. However, there are lots of details to be taken care of for them. For example, the former method may increase the extra index size greatly while the second method may not be directly suitable to a proximity-aware retrieval system unless we allow random lookups within the resulting lists of all two-term queries. In addition, it is interesting to study how to integrate the more optimal index compression methods to decrease the index sizes, e.g., PForDelta [33] which has been shown to be efficient in both compression size and decompression speed. It will also be interesting to see if we should reorganize the extended indexes themselves such that the early termination inside them is. Finally, we want to study the impacts on our methods of other factors, such as query features, caching policies and user feedbacks.

7. REFERENCES

[1] V. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In Proc. of the 24th Annual SIGIR Conf. on Research and Development in Information Retrieval (SIGIR'01), 2001.

[2] V. Anh and A. Moffat. Pruned query evaluation using pre-computed impact scores. In Proc. of the 29th Annual SIGIR Conf. on Research and Development in Information Retrieval (SIGIR'06), 2006.

[3] D. Bahle, H. E. Williams, and J. Zobel. Efficient phrase querying with an auxiliary index, In Proc. of the 25th Annual SIGIR Conf. on Research and Development in Information Retrieval (SIGIR'02), New York, NY, USA,2002.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In Proc. of the 7th Intl. Conf. on World Wide Web (WWW'98), 1998.

[5] A. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In Proc. of the 12th Conf. on Information and Knowledge Management (CIKM'03), Nov 2003.

[6] S. Büttcher and C. Clarke. A document-centric approach to static index pruning in text retrieval systems. In Proc. of the

15th ACM Intl. Conf. on Information and Knowledge Management (CIKM'06), 2006.

[7] S. Buttcher and C. Clake. Efficiency vs. effectiveness in terabyte-scale information retrieval. In Proc. of the 14th Text Retrieval Conference (TREC'05), 2005.

[8] S. Buttcher and C. Clarke, B. Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In SIGIR'2006.

[9] Carnegie Mellon University, The ClueWeb09 Data set, 2009, http://boston.lti.cs.cmu.edu/Data/clueweb09/

[10] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. JCSS, 66(4):614–656, 2003.

[11] R. Fagin. Combining fuzzy information from multiple systems. JCSS, 58(1):83–99, 1999.

[12] R. Fagin. Combining fuzzy information: an overview. *SIGMOD Rec.*, 31(2):109-118, 2002.

[13] U. Güntzer, W. Balke, and W. Kiebling. Optimizing multi-feature queries for image databases. In Proc. of the 26th Intl. Conf. on Very Large Data Bases (VLDB'00), pages 419–428, 2000.

[14] M. Kaszkiel, J. Zobel, and R. Sacks-Davis. Efficient passage ranking for document databases. ACM Transactions on Information Systems, 17(4):406–439, Oct. 1999.

[15] O. Kretser, A. Moffat. Effective document presentation with a locality-based similarity heuristic. In Proc. of the 22th Annual SIGIR Conf. on Research and Development in Information Retrieval, 1999

[16] R.Kumar, K. Punera, T. Suel and S. Vassilvitskii, Top-*k* aggregation using intersections of ranked inputs, In Proc. of the Second ACM Intl. Conf. on Web Search and Data Mining, 2009

[17] X. Long and T. Suel. Optimized query execution in large search engines with global page ordering. In Proc. of the 29th Intl. Conf. on Very Large Data Bases, September 2003.

[18] X. Long and T. Suel. Three-level caching for efficient query processing in large web search engines. In Proc. of the 14th Intl. Conf. on World Wide Web, pages 257–266, 2005

[19] A. Moffat and J. Zobel. Fast ranking in limited space. In Proc. of the 10th IEEE Intl. Conf. on Data Engineering. Houston, TX, February 1994.

[20] E. de Moura et al. Improving web search efficiency via a locality based static pruning method. In Proc. of the 14th Intl. Conf. on World Wide Web, pages 235–244, 2005.

[21] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. JASIS, 47(10):749–764, 1996.

[22] Y. Rasolofo and J. Savoy. Term proximity scoring for keyword-based retrieval systems. In Proc. of the 25th European Conf. on IR Research, pages 207–218, April 2003.

[23] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In Proc. of the 3rd Text Retrieval Conference (TREC), Nov 1994.

[24] R. Schenkel, A. Broschart, S. Hwang, M. Theobald and G. Weikum. Efficient text proximity search. In Proc. of the 14th String Processing and Information Retrieval Symposium, 2007.

[25] G. Skobeltsyn, F. Junqueira, V. Plachouras, R. Baeza-Yates: ResIn: a combination of results caching and index pruning for high-performance web search engines. In Proc. of the 31st Annual SIGIR Conf. on Research and Development in Information Retrieval, 2008.

[26] R. Song, M. Taylor, J. Wen, H. Hon, Y. Yu. Viewing term proximity from a different perspective. vol 4956, pp. 346-357, Springer Berlin / Heidelberg , 2008.

[27] T. Strohman and W. Croft, Efficient Document Retrieval in Main Memory, In Proc. of the 30th Annual SIGIR Conf. on Research and Development in Information Retrieval

[28] T. Tao and C. Zhai, An exploration of proximity measures in information retrieval, In Proc. of the 30th Annual SIGIR Conf. on Research and Development in Information Retrieval.

[29] Y. Tsegay, A. Turpin, and J. Zobel. Dynamic index pruning for effective caching. In Proc. of the ACM 16th Conf. on Information and Knowledge Management, 2007

[30] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. Information Processing and Management, 31(6):831–850, 1995.

[31] I. Witten, A. Moffat, and T. Bell. Managing gigabytes: compressing and indexing documents and images. Morgan Kaufmann, second edition, 1999

[32] H. Yan, S. Ding and T. Suel. Compressing term positions in web indexes, In Proc. of the 32nd Annual SIGIR Conf. on Research and Development in Information Retrieval (SIGIR'09), Boston, July, 2009

[33] H. Yan, S. Ding and T. Suel, Inverted Index Compression and Query Processing with Optimized Document Ordering, The 18th Intl. World Wide Web Conference (WWW'09), Madrid, Spain, April 2009

[34] F. Zhang, S. Shi, H. Yan and J. Wen. Revisiting globally sorted indexes for efficient document retrieval. In Proc. of the Third ACM Intl. Conf. on Web Search and Data Mining (WSDM'10), 2010.

[35] M. Zhu, S. Shi, M. Li, and J.-R. Wen. Effective top-K computation in retrieving structured documents with term-proximity Support. In Proc. of the ACM 16th Conf. on Information and Knowledge Management (CIKM'07), Portugal, 2007.

[36] M. Zhu, S. Shi, N. Yu, J.-R. Wen. 2008. Can phrase indexing help to process non-phrase queries? In Proc. of the ACM 17th Conf. on Information and Knowledge Management (CIKM'08), 2008

[37] J. Zobel and A. Moffat. 2006. Inverted files for text search engines. ACM Computing Surveys. Vol. 38, No 2, Jul. 2006.