Heuristics to Improve the BMW Method and Its Variants

Lídia Lizziane Serejo de Carvalho, Edleno Silva de Moura,
Caio Moura Daoud, Altigran Soares da Silva

Universidade Federal do Amazonas, Brazil
lidializz@gmail.com, {edleno, caiodaoud, alti}@icomp.ufam.edu.br

Abstract. In this paper, we propose and evaluate heuristics to improve the performance of BMW and its variants. The proposed changes maintain the property of preserving the order of the top ranking results, while reduce query processing times and the amount of memory required for processing queries.

Categories and Subject Descriptors: H.3 [**Information Storage and Retrieval**]: Miscellaneous

Keywords: Information Retrieval, Query Processing, Inverted Indexes, Search Engines

1. INTRODUCTION

Query processors in search engines are implemented aiming to minimize costs with respect to the response time and memory usage. Their main goal is to take a query specified by the user and provide a ranking with the documents considered relevant to such query. To do so, they use Information Retrieval models that produce ranking functions, like the vector model [Salton et al. 1975] and the BM25 model [Robertson and Walker 1994], to compute a weight for each document given a query. In general, the systems return a ranking with the top-k documents with the higher weights, where k is a predefined number.

Ding and Suel [2011] proposed the Block-Max WAND (BMW) algorithm, an algorithm for processing queries that proved to be faster than the previously proposed methods for computing top-k ranking results. Variants of BMW method have been proposed and studied since then, including the BMW-CS [Rossi et al. 2013], which reduces the query processing times when compared to BMW, but has the disadvantages of increasing the amount of memory required for processing queries and the possibility of losing results, thus changing the list of documents that appear at the top of the response for each query.

Algorithms such as BMW use document discarding techniques (pruning techniques) that are directly related to the current heuristics that aim at reducing the processing costs. BMW maintains a minimum heap structure that stores the top-k documents during query processing. The smallest weight among all the documents present in the heap is used as a discarding threshold. Thus, while processing a query, only the scores of documents that have a chance to overcome the discarding threshold are computed.

The main objective of this work is to study and propose improvements to the BMW algorithm and its variant BMW-CS in order to reduce the query processing times and the amount of memory necessary to process queries with these algorithms. The most remarkable results are as follows: (i) reduction in the query processing times with an initial threshold pruning heuristic. (ii) reduction in memory usage due to a significant reduction in the size of an auxiliary structure used by the algorithms

when processing queries, the skiplists. In both cases, the changes made have the advantage of not changing the initial results provided by the methods (BMW or BMW-CS). As BMW is an algorithm that preserves the top results for each query, our changes on the BMW also hold such property.

The remainder of this paper is structured as follows. The Section 2 presents the summary of the main related research. In Sections 3 e 4, present our proposals. The Section 5 describes the experiments and the obtained results. Finally, in Section 6, we discuss the conclusions and directions for future research.

2. RELATED WORK

Several researchers have proposed strategies to improve the performance of query processing in search engines. The query processing methods found in the literature can be divided into two main categories: those that use TAAT (Term-At-A-Time) strategy and those that use DAAT (Document-At-A-time) strategy [Baeza-Yates and Ribeiro-Neto 2011]. In methods that use TAAT strategy, the inverted lists are sorted by the weight of the documents. Thus, the query is processed one term at a time. The biggest disadvantage is that this strategy requires large amounts of memory to store the scores achieved by each document for each term in the query. In the methods using DAAT strategy, the inverted lists are sorted by the *id* of the document, allowing the lists to be traversed in parallel, and the full score of a document can be computed at each iteration. Another feature of this strategy is the use of *skiplists* associated with each inverted list. These structures store pointers to entries in the inverted list, dividing it into blocks to facilitate access to documents, which typically have between 128 and 256 entries [Broder et al. 2003].

One of the most important works presented in recent years proposes a DAAT query processing method known as WAND (Weak AND or Weighted AND) [Broder et al. 2003]. In this method the highest score of each inverted list is stored during the document collection indexing. Knowing the maximum score that a document can achieve is important to avoid the cost of computing scores of documents that have no chance to change the top of ranking results. At query processing time, documents that are being evaluated, called pivots, are analyzed in two stages. First, the Max Score is obtained, maximum score of each list where the pivot has a chance to occur. When the Max Score is greater than the current pruning threshold, the inverted list of each term will be accessed so that the pivot document has its real score computed. When the Max Score does not exceed the pruning threshold, the pivot document is discarded and a new pivot is selected.

The BMW algorithm [Ding and Suel 2011] was created based on the WAND [Broder et al. 2003], but proposes an optimized solution that mainly covers a modification of skiplists. In BMW, for each input of skiplists, the highest score of the block is stored. The pruning strategy is similar to that adopted in the WAND method, with the advantage of a maximum score closer to the actual score of each document. With this advantage, the BMW method discards even more documents that do not have sufficient weight to be inserted on top of answers ranking. Pruning in the BMW occurs in two stages, (i) when the *Max Score* does not overcome the discarding threshold, as in the WAND method, and (ii) when the *Block Max Score* does not overcome the discarding threshold. The Block Max Score is computed based on the maximum score of the blocks in which the document can occur and is only checked when the Max Score exceeds the discarding threshold.

Discarding threshold is dynamically updated whenever a document has score higher than the current threshold. This document updates the heap of answers and the pruning threshold is updated with the weight of the smallest document in the heap.

For illustrative purposes, assume a scenario as shown in Figure 1. The pruning threshold indicates that it is necessary a document with a score higher than 3.2 to upgrade the current answers ranking. The query processing algorithms keep an invariant that any list pointing to a document at each query processing step does not contain documents that are smaller than it and pointed by other lists at the

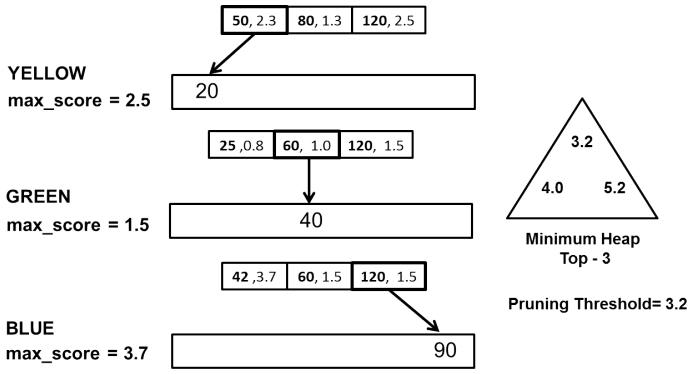


Fig. 1. Inverted lists for the query terms "yellow green blue".

same step. The figure describes a state of the processing were documents 20, 40 and 90, associated with each query term "yellow green blue" will be evaluated, in that order. The inverted list of the first document to be processed has a maximum score of 2.5, i.e., the document 20, which only occurs in the list of term "yellow", does not have sufficient weight to overcome the pruning threshold. However, the document 40, which can occur in the inverted list of the term "yellow", may have a score up to 4.0 ($2.5 + 1.5 > 3.2$), higher than the current pruning threshold. Since such a possibility exists, the algorithm checks the Block Max Score of the pivot (document 40). If the pivot occurs in the inverted list of the term "yellow" and it has the highest score among all documents of its block, on both lists, its maximum score will be 3.3 ($2.3 + 1.0 > 3.2$), therefore higher than the pruning threshold. After those verifications, the document 40 has its full score evaluated. In this example, the complete score of the document is 3.3 ($2.3 + 1.0$), which exceeds once more the threshold pruning, so the document 40 is added to heap of answers, updating the pruning threshold. At this stage, the pointers of the inverted lists are moved so that they point to the next document higher than the evaluated pivot, and then the algorithm proceeds repeating the process.

Some authors explore the partitioning approach of the inverted list in more than one tier to speed up query processing. One of the most recent, and also more successful partitioning strategy, is adopted by the BMW-CS algorithm [Rossi et al. 2013], which divides each inverted list into two tiers. The first tier has smaller size and is created using inputs that have the highest scores among the ones of each inverted lists, while the second tier has a larger index and contains the remaining entries.

The query processing is divided into two stages. The first, called the Candidate Selection, processes the first tier of the lists of the query terms. At this stage, the process is similar to the method used in the BMW, but also considers the maximum score of the document on the second tier. The value for the second tier is important, because there can be documents that are in the first layer, but also occur in the second layer to other query terms. The documents selected in the first stage are called candidates to be included in the answer. In the second stage, the second tier is traversed only to find occurrences of candidate documents in the second tier and compute the real score of such documents.

The BMW-CS method has the disadvantage of not preserving all the top-k answer results of a query, as documents that doesn't occur in the first tie of at least one of the query terms are not evaluated. In these cases, even if the documents have sufficient scores to be inserted into the answer, they will be discarded by the BMW-CS algorithm.

3. STUDY OF INITIAL THRESHOLDS FOR PRUNING

The pruning threshold is crucial for document discarding strategies to work correctly in the presented methods. At first, while the heap containing the top results is not full, the pruning threshold is

zero. This means that, in the beginning, the pruning strategies are not applied by the BMW and its variants, since any considered document has a score higher than zero. After processing the first k documents, the methods obtains the first pruning threshold different from zero, starting the document discarding process that reduces the cost of query processing in the BMW algorithm and its variants. As more documents are processed, the pruning threshold increases until achieve its maximum value, which is the minimum value among top results of the final ranking.

The adoption of a low initial threshold slows down the BMW algorithm and its variants at the initial stage of query processing, since fewer documents are discarded at the beginning. Our first attempt to improve the algorithms was to study the performance of such algorithms when using as initial pruning threshold an estimated value that is both safe, guaranteed to be smaller than the maximum possible pruning threshold obtained at the end of processing, and large enough to impact the processing costs, having the closest possible value to the final pruning threshold.

The strategy for adoption of an initial threshold was applied in two scenarios: When we are interested in computing the top-10 results and the top-1000 for a given query. The two scenarios are considered typical in search systems and are usually studied in articles that deal with efficient query processing [Rossi et al. 2013]. To estimate the initial pruning threshold, we keep, during the indexing, the k-th highest score of each word of the collection in its corresponding entry in the inverted index structure. Given a query, it is estimated an initial minimum score value by taking the highest k-th score within the query terms. This choice ensures that the initial threshold will not discard answers that belong to the top k results, because it is known that there are at least k entries with a score at least equal to the chosen initial threshold. On the other hand, the adoption of this threshold allows to discard a priori entries that would have been taken into consideration when adopted an initial threshold equals zero.

In addition to propose strategies and try to determine an initial threshold, we also verified how much we can improve the performance with this type of strategy. To do this, the minimum score among k responses obtained from the query processing was stored. Then, the same queries were processed again, but starting the pruning threshold with its maximum value achieved by the end of query processing. We call this value as optimal threshold for being indicative of the highest gain we could get with the use of the initial pruning threshold strategy. The optimal threshold is only a reference value, since to obtain it we need to first process the query, achieve the top k scores and then select the smaller among then as the optimal threshold.

Finally, it is important to say that the heuristic of taking the highest k-th score of each inverted list to set the initial threshold can be implemented at a very small (and not relevant) extra cost at indexing time. The current algorithms, such as BMW and BMW-CS, already require the computation of the maximum score, and the block max scores of each list. The k-th score can be computed in the same procedure.

4. VARYING THE BLOCK SIZES

In the experiments presented by Ding and Suel [2011] and Rossi et al. [2013], fixed blocks of 128 documents were used in the creation of the skiplists. This means that for each 128 documents in an inverted list, an entry is created in the skiplist to accelerate the documents access and store discarding information, such as the highest score of each block (Block Max Score). As seen in the Figures 4(a), 4(b) and in the Table I, the smaller is the block size, the shorter are the query processing times. Whenever decoding a block, the system should decode its whole content, which means the cost of accessing documents increase with the block size adopted.

Whenever the Max Score and the Block Max Score of the pivot overcome the pruning threshold, the corresponding blocks will be accessed seeking the pivot document so that its real score is computed. Blocks are totally discarded when the Block Max Score of the pivot does not exceed the pruning

Table I. Performance of BMW algorithm for fixed size blocks of 64, 128 and 256 documents. The number of blocks is represented by *.

		Pivots	Time(ms)
Blocks of 64	Top-10	369.781.344	20
23.595.719*	Top-1000	1.002.528.576	51
Blocks of 128	Top-10	426.335.552	22
11.798.404*	Top-1000	1.116.841.472	55
Blocks of 256	Top-10	495.183.424	27
5.899.760*	Top-1000	1.223.783.040	59

threshold. Thus, blocks with low Block Max Score have more probability to be discarded. In this case, we infer that adjacent blocks which have low value of Block Max Score could be integrated, in an attempt to discard more documents when processing the queries.

We adopted the 1000-th highest score of each term in the collection as a reference to determine whether the Block Max Score is low or not, making an association with the typical amount of inputs usually considered in the search results. Therefore, blocks with Block Max Score smaller than the reference value are considered here as blocks of low impact that can be merged with other blocks.

The Figure 2 illustrates an inverted list for a term A which is divided into n blocks, each one composed of 128 documents. In the example, the 1000-th highest score of the list has already been computed and has value equal to 12. In this scenario, the blocks A_2 and A_3 are combined because they have maximum score smaller than the reference value 12. The blocks A_1, A_4 and A_n remain as they were. The new block $A_2 + A_3$ will have 256 documents and Block Max Score equal to 8.

Our initial hypothesis was that the heuristic of block union described above could be used to improve the pruning process in search algorithms, such as BMW and BMW-CS. In the Experiments section, we show that our initial hypothesis was not valid, but still the use of variable block sizes has an interesting positive effect, since it reduces the memory requirements when processing queries.

The gain in performance with variable blocks was not effective due to negative side effects that occur when merging distinct blocks. We analyzed some queries in order to identify which blocks were accessed during the stage of query processing, and then create an index with a structure where adjacent blocks, not accessed, were aggregated into a single block. The objective was to handle queries to create a block structure where we would know *a priori* which blocks would be discarded during the query processing, and try to use this information to maximize the blocks merging. Notice that, as the optimal threshold, this strategy requires a previous evaluation of the query, thus being not viable in practice. At the end, we would have an indication of the time in which a query would be processed when applying the strategy of varying blocks size.

The Figure 3 gives an example to illustrate the possible impact of joining blocks in inverted lists. At a first moment (Figure 3(a)), the lowest document of the answers heap presents score equal to 14, and the selected pivot is the document 3097. The document 3097 is present in the inverted lists of the terms of A, B and C, and we have the value of the scores of each block to which it belongs. Since the sum of the maximum score of the block $(3.6 + 6.4 + 3.3)$ of all occurrences of the pivot document

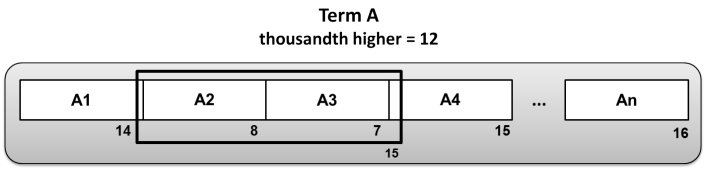


Fig. 2. Example of the inverted list with variable blocks.

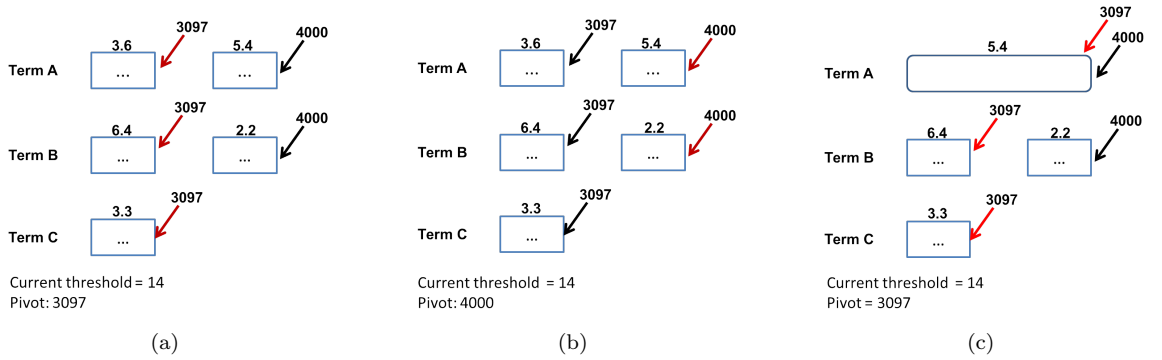


Fig. 3. An example to illustrate the possible negative impact of joining adjacent blocks in inverted lists.

is not higher than the minimum heap (current threshold), a new pivot will be selected. When the new pivot is selected (document 4000), the sum of the maximum score of block in all instances of this document in the index also does not surpasses the current threshold. The algorithm then keeps searching for possible candidates to be returned as response.

In this example, the blocks which contain the document 3097 and 4000, for the term A, were not read, but the remaining blocks have been accessed. To handle this query, we join the blocks that have not been accessed, because we intuitively think that the algorithm may skip more documents (blocks formed by more entries), and thus reduce the query processing costs. The Figure 3(c) illustrates this new scenario where the query "A B C" presents a structure with variable blocks. At this point of the query processing, the pivot document 3097 will be evaluated. With the block structure modified, the total value of maximum score is 15.1 (5.4 + 6.4 + 3.3). This value is higher than the current threshold and indicates that the pivot document will have its complete score computed, i. e., the algorithm will scan a larger block looking the pivot document, which will affect its performance.

The heuristics to adopt variable block sizes require an extra step at indexing time. Our current implementation uses the index already constructed with fixed block sizes to create the version with variable sizes. This process adds, in the current version, less than 10% of the total time to build the index, but the process can be optimized in the future. While this represent an extra cost, we can say that the strategy of variable block sizes does not add a prohibitive extra cost at indexing time.

5. EXPERIMENTAL EVALUATION

For the experiments, we used and modified the search system implemented by Rossi et al. [2013]. The adopted system processes queries using either the algorithm BMW or the BMW-CS variant. We adopted the TREC GOV2 reference collection for the experiments [Clarke et al. 2004]. This consists of 25.205.179 million web pages crawled from the .gov domain in early 2004. The TREC GOV2 has 426 GB of text, composed of HTML pages and text extracted from pages in PDF and Postscript (PS) format. The full index has about 7 gigabytes of inverted lists and a vocabulary of about 4 million distinct terms. The TREC GOV2 collection was adopted due to the fact that it became a standard for studies that propose efficiency improvements in search engines, having been used in the experiments of the main studies found in the literature.

A set of 10.000 queries extract was randomly selected from the TREC 2006 efficiency queries. All stop-words of these queries have been removed. During the query processing, the entire index is loaded to memory in order to avoid any possible bias in the query processing time. All of these setup options were chosen for being similar to those adopted in previous studies [Strohman and Croft 2007; Ding and Suel 2011], which makes it easier to compare the studied methods. We ran the experiments at an Intel(R) Xeon(R), with X5680 Processor, 3.33GHz and 64GB of memory. All the experiments

were performed in scenarios where the top-10 or top-1000 answers were returned. The recovery of the top-1000 answers was included to simulate an environment where the set top-1000 is used as input to a more sophisticated ranking method, as for example using a machine learning technique. The scenario of the top-10 answers was included to simulate a more usual scenario, where the user is only interested in a small list of results for his or her query. The algorithms were evaluated in terms of processing time and memory used.

The strategies were implemented in the BMW algorithm proposed by Ding and Suel [2011] and the BMW-CS algorithm proposed by Rossi et al. [2013]. These were developed in C++. Just as the original algorithms, we continued to use the probabilistic model Okapi BM25 as similarity function. For the BMW-CS algorithm, we varied the first tier size from 2 to 50 percent of the full index.

5.1 Results With Pruning Initial Threshold

To estimate the initial pruning threshold, we keep, during the indexing, the k-th highest score of each word of the collection in its corresponding entry in the inverted index structure. Given a query, it is estimated an initial minimum score value by taking the highest k-th score within the query terms. This choice ensures that the initial threshold will not discard answers that belong to the top k results, because it is known that there are at least k entries with a score at least equal to the chosen initial threshold. On the other hand, the adoption of this threshold allows to discard a priori entries that would have been taken into consideration when adopted an initial threshold equals zero.

The Figures 4(a) and 4(b) show gains by using the k-th highest score of each word of the collection to select the initial threshold. The initial threshold is chosen by taking the bigger k-th highest score among the words present in the query. This choice guarantee that there is no chance of changing the ranking, since we know that at least k documents in the list from where the threshold was selected have score higher than the initial threshold. In these two figures, we compare the proposed strategies to BMW algorithm without initial pruning threshold (Original BMW) for the top-10 and top-1000 query results, respectively. The figures also present the results of varying the size of the blocks generated. Experiments were executed with fixed block sizes of 64, 128 and 256 documents. Fixed block sizes of 64 documents result in smaller query processing time, but as indicated in the Table I, they form more blocks in the inverted index, requiring more memory to store the skiplists. In the opposite way, fixed block sizes of 256 documents form fewer blocks, however result in higher query processing times. The larger the amount of formed blocks (number of entries in the skiplist structure), the high is the cost of memory used when processing queries, as more space is necessary for keeping the skiplists in memory.

The block size typically used in studies reported in the literature is of 128 entries. With this configuration, it is observed that the choice of an optimal initial threshold results in a gain of about 18% in the query processing times for the top-10 answers and 16% for the top-1000 answers, which would be the limit for any initial threshold estimation strategy. As can also be observed in the Figure 4, our heuristic of estimating an initial threshold also impacted the times when computing top-1000 results, with a reduction of approximately 5.5% when compared to the original algorithm. The major gains occurred with blocks sized of 64 documents, where we had reductions of approximately 5.2% and 7.8% for top-10 and top-1000, respectively.

When looking at the experiments with BMW-CS algorithm, for fixed blocks of 64, 128 and 256 documents, and in the two tiers, the higher impacts of the method with an initial threshold strategy are obtained for top-1000 answers. Within this scenario, we calculated the average time between the variations of 18% and 50% of the first tier for all implementation options, as shown in Table II. The table shows that the introduction of the thousandth score as the initial threshold reduces the query processing times by up to approximately 17%.

The Figures 5(a) shows time performance achieved when varying the first tier size and adopting the initial threshold strategy and fixed block sizes of 64, 128 and 256. Complementary, Figure 5(b) shows

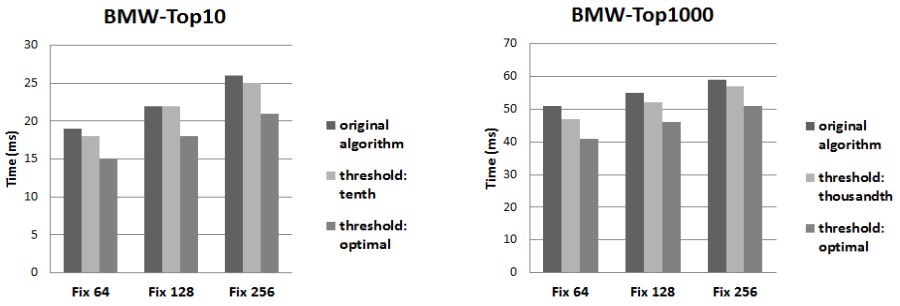


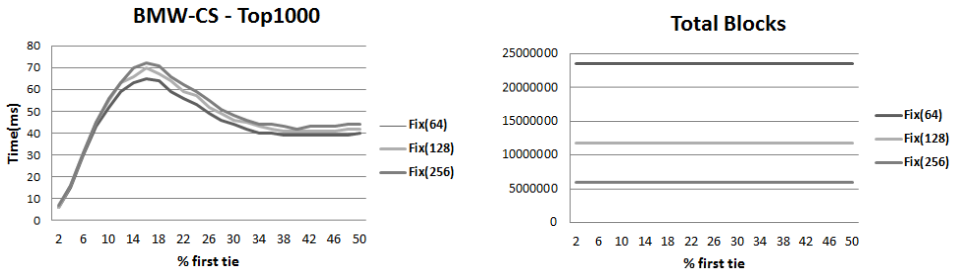
Fig. 4. Performance of BMW algorithm on different approaches to initial pruning thresholds: Original BMW, BMW with tenth/thousandth highest score and BMW with the minimum score of the top-k answers to fixed blocks of 64, 128 and 256 documents.

Table II. Average of the processing time in milliseconds calculated from the variation of 18% of the first layer, for top-1000 answers.

	Fixed 64	**Fixed 128**	**Fixed 256**
Original BMW-CS	54,65	57,94	59,94
BMW-CS with threshold	45,12	47,82	49,88
Reduction (%)	17,44	17,46	16,78

the evolution in the total number of blocks in these three scenarios, again when varying the first tier sizes. It can be noticed that there are not much oscillation in the total of blocks as we increase the size of the first tier. Notice also that the use of fixed blocks of 64 entries implies in significant increasing in the total number of blocks, which also means increasing gin the size of the skip lists to address these blocks, and thus an increasing in the memory requirements. Given that the adoption of blocks of size 64 does not cause significant improvement in time performance when compared to the adoption of sizes 128 and 256 (as shown in Figure 5(a)), we can conclude that the adoption of blocks sizes of 64 entries should be discarded. The adoption of blocks of size 64 in the experimented collection would result in about 23.6 thousand blocks, while the adoption of block sizes of 128 and 256 documents would result in about 12 thousand and 6 thousand blocks, respectively. Therefore, as is shown in Figure 5(a), When comparing the options of block sizes 128 and 256, again there is an advantage for the option of size 256, since again it does not cause a high degradation in the time performance, but significantly reduces the total number of blocks in the index, thus allowing a significant reduction in the memory required to maintain the skiplists.

As already discussed in this section, the reduction in query processing times achieved when applying the initial threshold for the BMW-CS algorithm was best seen when computing top-1000 answers. To



(a) Query processing with initial threshold (b) Total entries in the skiplists

Fig. 5. Performance and total fixed blocks of the BMW-CS algorithm.

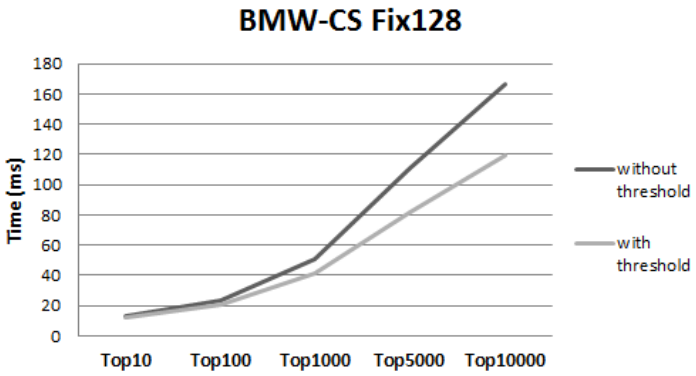


Fig. 6. Performance of the BMW-CS algorithm for fixed blocks when varying the number of positions preserved in the ranking from top-10 to top-10000. The first tier size was fixed to 40% of the index.

better understand this behavior, Figure 6 shows the time performance of BMW-CS algorithm with fixed block sizes of 128 entries and with the first tier formed by 40% of the index. Exceptionally, this experiment involves other scenarios besides the top-10 and top-1000 answers returned in the query processing. For this one, the processing time for top-10, top-100, top-1000, top-5000 and top-10000 documents were recorded. By analyzing the graphic we can infer that the higher the amount of returned results (top) for each query, the larger is the impact of the insertion of the initial threshold. The explanation for this phenomenon is that, as the number of top results to be preserved increases, as longer the heap of results takes to reach its final configuration, and thus reach the minimal threshold. Thus, strategies that estimate the threshold are likely further reduce the query processing times in scenarios where the number of top elements to be preserved is high.

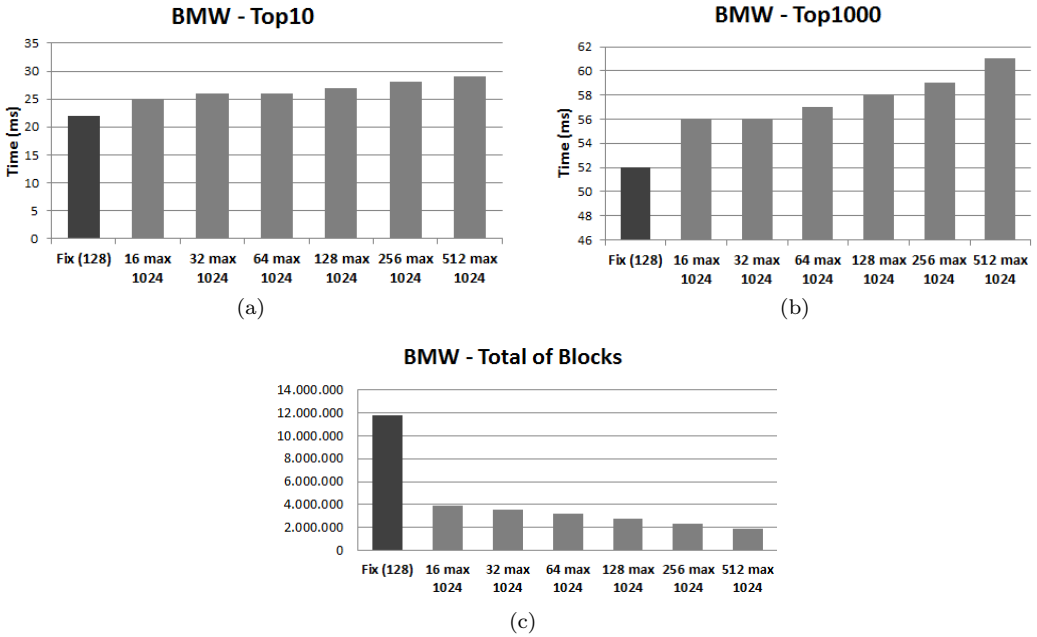


Fig. 7. Performance of the BMW algorithm with the use of the approaches of variation of the blocks size (maximum of 1024 documents).

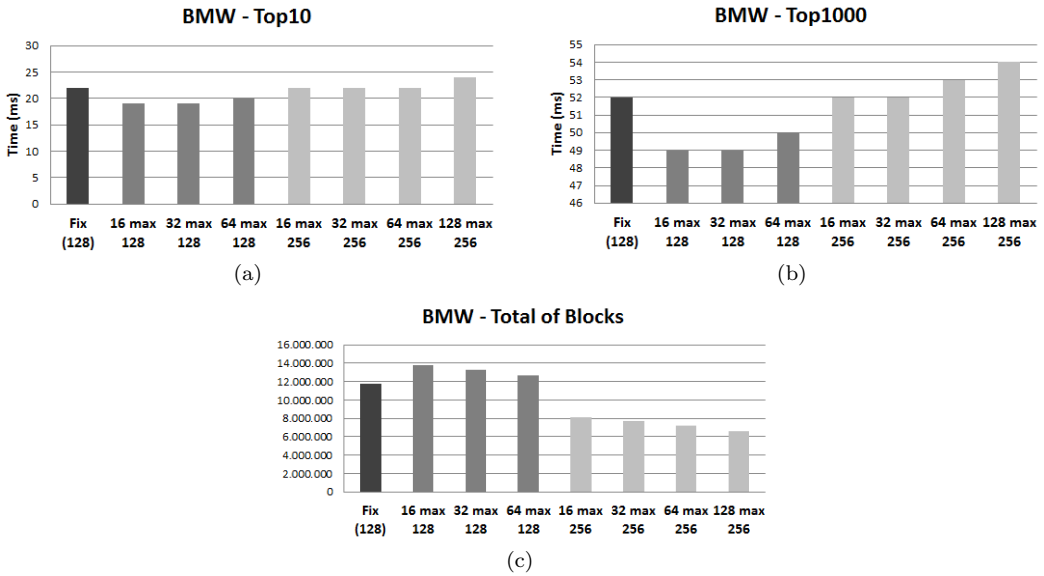


Fig. 8. Performance of the BMW algorithm with the use of the approaches of variation of the blocks size (maximum of 128 and 256 documents).

5.2 Varying the Size of the Blocks

We started by evaluating the impact of varying the size of the blocks in the BMW. As a first strategy, we set a minimum and maximum size for the blocks. We varied blocks of at least 16, 32, 64, 128, 256 and 512 documents, composing the maximum of 1024 documents. The Figure 7 shows a comparison of this strategy with the Original BMW algorithm. We observed that the query processing time increases for both top-10 and top-1000 scenarios. When computing top-1000 results, and applying the variation with 512 blocks up to 1024 documents, the time increases from 52 milliseconds to a little over 60 milliseconds. On the other hand, we also see from the Figure that, despite the increase in query processing time, by using the strategy we managed to decrease to approximately 80% of the total number of blocks formed (Figure 7(c)) when compared with the Original BMW algorithm.

Another strategy we have experimented was to reduce the maximum size of 1024 to 128 or 256 documents. The graph presented in the Figure 8 shows the results achieved with these new variations. There is a reduction in query processing times for indexes formed by blocks with a maximum of 128 documents for the top-10 and top-1000 answers. Moreover, it can be noticed that we can maintain similar processing time to the Original BMW when using the maximum size of 256 documents per block. In Figure 8(c) it can be seen that the total number of blocks formed was reduced by about 44%. For instance, generating blocks with minimum size of 32 and maximum of 256 documents resulted in the same query processing time obtained by using the Original BMW (52 milliseconds). However, the number of blocks generated by this choice was 35% lower, generating 7.676.292, against 11.798.404 blocks from the Original BMW. This is a significant reduction in the amount of memory to store information generated for each block.

As discussed in Section 4, changes in the size of the blocks can produce side effects on the performance of the algorithms, as they also change the maximum score values of the blocks and, consequently, affect also the document pruning strategies, either on the BMW and BMW-CS algorithms. Nonetheless, we may achieve a reduction in the total number of blocks formed without having a negative impact on query processing times. Bringing together the use of variable size blocks and the initial threshold insertion strategy, we obtained gains in processing time and, at the same time, reduction in the size of the skiplists used to represent documents blocks when processing queries.

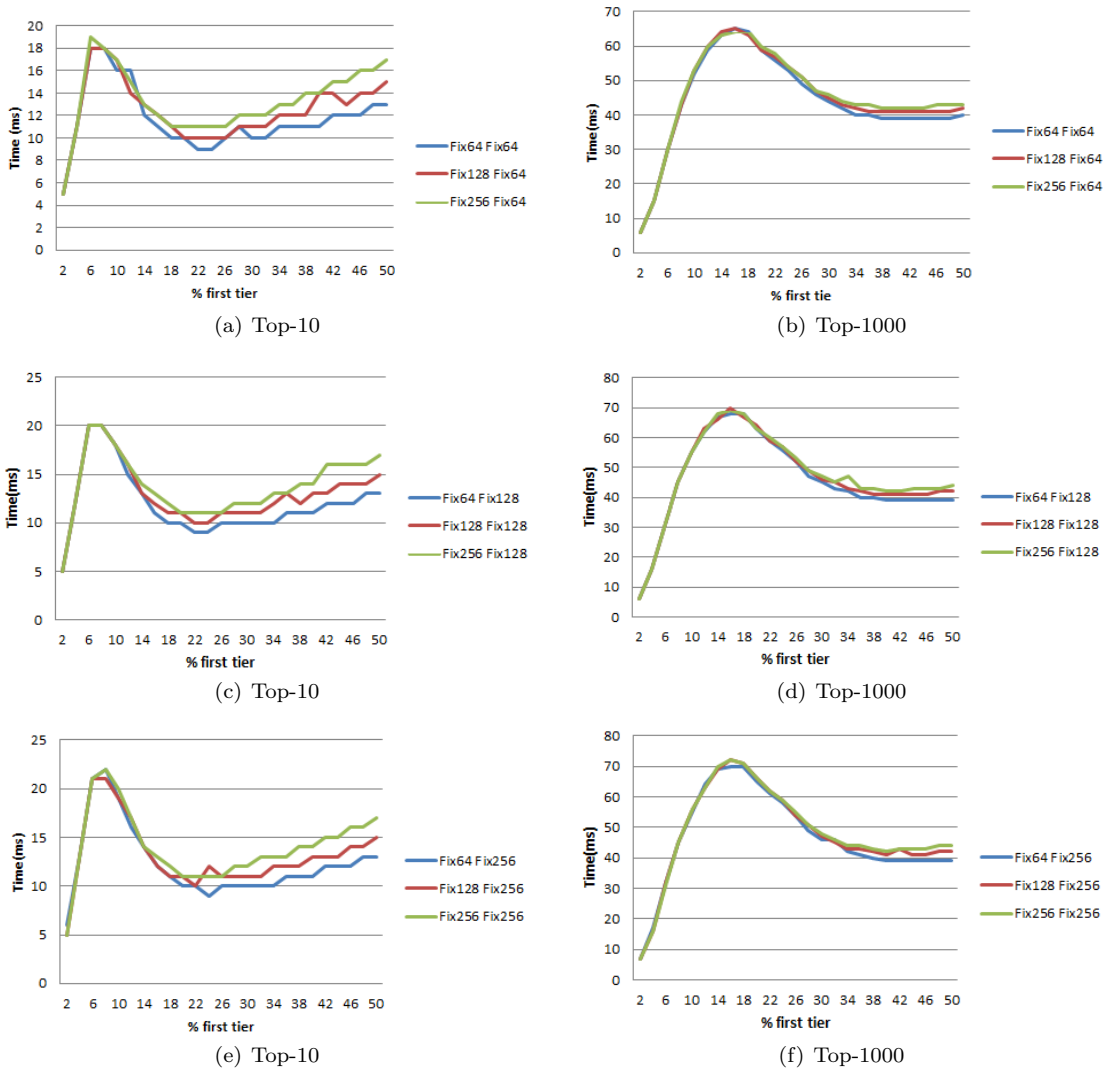


Fig. 9. Performance of the BMW-CS algorithm varying the number of fixed blocks in the two tiers of the index.

Another option to vary the block size is adopt fixed sizes on each tier, but allowing a difference in size between the tiers. Experiments with this scenario are presented in Figure 9, where we experimented with distinct sizes (64, 128 and 256), in the two tiers of the inverted list in the BMW-CS algorithm. All options presented adopt the initial pruning threshold strategy also proposed here. We present three variations on each chart, choosing those that best define the algorithm's behavior when using fixed size blocks. It is important to remember that, like BMW, the smaller is the block size, the higher is the number of entries in the skiplists. This indicates that there will be larger memory requirements whenever we reduce the block sizes.

The Figures 9(a), 9(c) and 9(e) indicate that blocks formed with fewer documents may reduce query processing times. However, when analyzing memory costs, we realize that fewer entries in the skiplists, i, e., blocks formed with more documents, generate a better setting for the algorithm. We also note that the differences in query processing times among all exposed variations (Figure 9) differ by a few milliseconds. This shows that we can reduce memory costs without causing much impact on the query processing times. In the worst case, where the block size is smaller, for example, 64 documents in

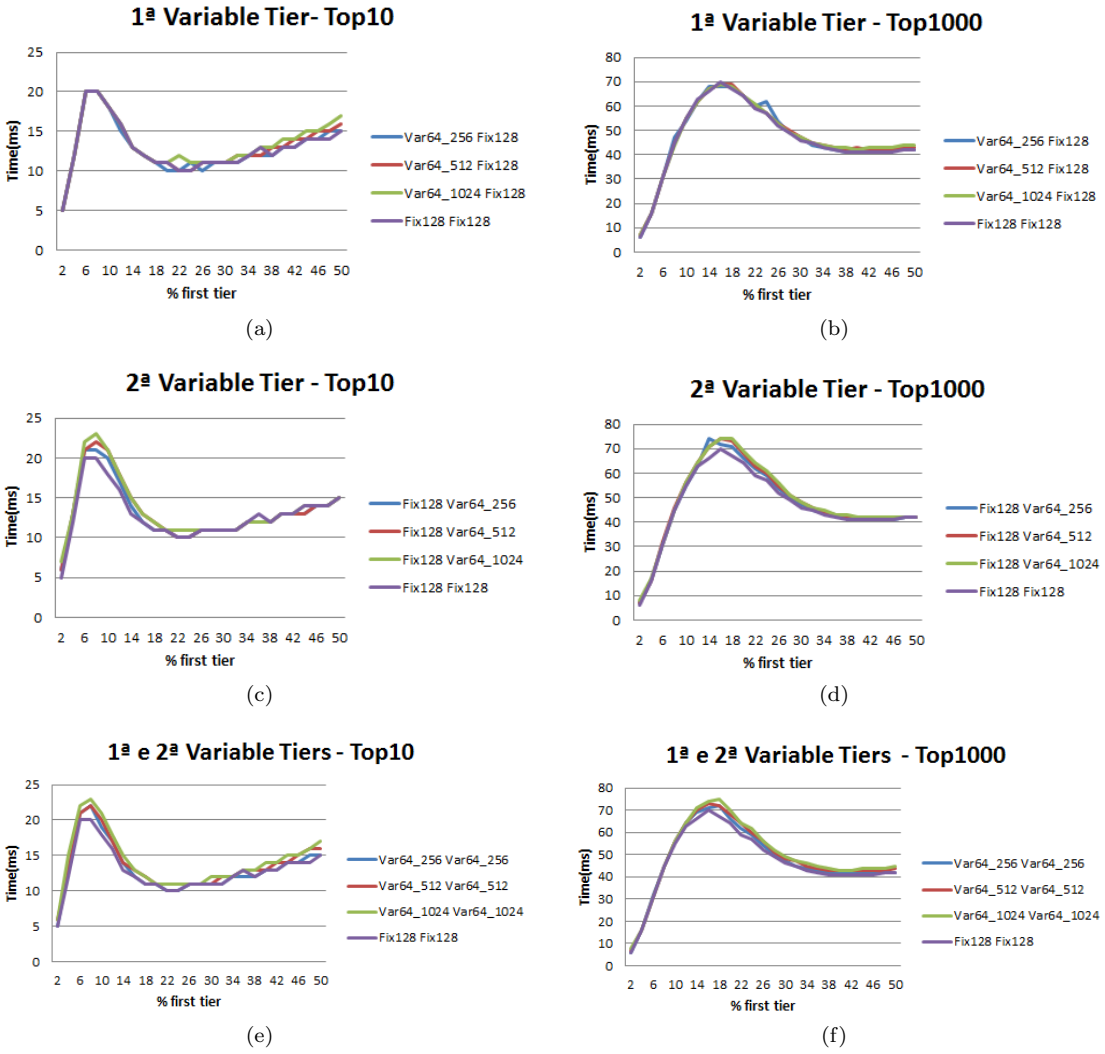


Fig. 10. Perfomance of the BMW-CS algorithm varying the number of blocks in the two tiers of the index.

both tiers, we have a total of about 24 million entries in the skiplists, while for blocks with 256, in both tiers, we have about 6 million entries. By implementing fixed blocks with 128 documents in the two tiers, we can achieve median costs in the total number of formed blocks (average of approximately 11 million) for each percentage of the first tier.

We also conducted experiments to study the application of variable size blocks in the BMW-CS algorithm. We varied the size of the blocks from 64 up to 1024 documents in both tiers and analyzed the algorithm performance for each percentage of variation of the first tier, to top-10 and top-1000 answers. All implementation options (with the first, second or both tiers variables) were compared to the BMW-CS algorithm using fixed blocks with 128 documents, and the initial threshold strategy proposed by us. The experiments (Figure 10) show that we did not obtain gains in query processing times for both the top-10 and top-1000 answers. The Table III shows the average time between variations of 18% to 50% of the first layer for all implementations options in the top-1000 answers scenario. It was found then that the query processing times obtained by the approaches are quite similar.

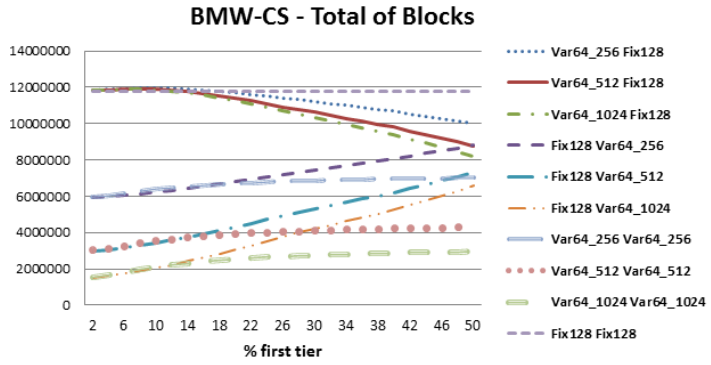


Fig. 11. Total number of blocks generated by the original BMW-CS and each of the alternatives of variable block sizes. Values are presented for a large range of first tier sizes. Bust curves are the ones with smaller value in the total number of blocks.

Table III. Variables Blocks - Average of the processing time in milliseconds calculated from the variation of 18% of the first layer.

	1ª variable tier	**2ª variable tier**	**1ª and 2ª variable tiers**
Min 64 Max 256	48,47	48,88	49,18
Min 64 Max 512	48,76	49,65	50,41
Min 64 Max 1024	49,00	50,12	51,35
Original BMW-CS: 57,94			
Original BMW-CS with threshold: 47,82			

Given that the query processing time was not affected by the strategies of variables block sizes, we investigated whether they result in reduction of memory usage or not, which can be measured by the total number of blocks generated when using each strategy. Figure 11 makes a comparison of the total number of blocks obtained when using each option of variable block size proposed by us. It can be noticed that BMW-CS algorithm with fixed blocks of 128 documents is the worst case of implementation among the ones experimented, resulting in about 11.8 million of blocks.

The implementation option of varying the two tiers with block sizes varying from a minimum of 64 and a maximum of 256 documents generates a reduction of about 43% on average when compared to the Original BMW-CS. Such results are obtained with just a small change in query processing times. For example, in our experiments, the average processing times to consider all size thresholds studied for the first tier (Table III) in the Original BMW-CS algorithm, that uses the initial threshold strategy, is of 47.82(ms), while we have an average time of 49.18 (ms) to vary the size of the blocks in two tiers with a minimum of 64 and maximum of 256 documents.

6. CONCLUSIONS AND FUTURE WORK

The results obtained show that the estimation of an initial threshold when discarding documents can improve the performance of the BMW algorithm and its BMW-CS variant for both the top-10 and top-1000 results scenarios. The most significant gain on modifying BMW for using on our initial pruning threshold strategy was achieved when the system uses blocks of fixed sizes 64, with a reduction of about 5.2% and 7.8% in query processing times for top-10 and top-1000 answers, respectively. Even greater gains were obtained when the modified version of the BMW-CS algorithm were used, with a reduction of approximately 17% in the query processing times.

The strategy of changing the size of the blocks of the index structure did not result in a reduction of query processing times, but provided significant decrease in the number of skiplists entries,

impacting then the amount of extra memory required by each method to process queries. For the BMW algorithm, the best configuration found in the experiments uses blocks with the maximum of 256 entries. In this case, the query processing times of the original algorithm were maintained, but the total number of blocks created decreased by about 35%. In the BMW-CS algorithm, the option of varying the size of the blocks in the two tiers, with a minimum of 64 and maximum of 256 documents, our experiments have shown an average reduction of 43% when compared to the original BMW-CS, without significant changes in the query processing times. These two options, combined with our proposed strategy of setting the pruning threshold to start as the value of the k-th highest score of the terms, are the best options proposed by us, according to the results of the experiments.

For future work, it would be interesting to study the impact of the techniques here implemented on scenarios where the search engine index does not fit in the main memory. In such situations, gains of space in the skiplists may become more relevant and have more positive impact in the query processing times, since small skiplists have a greater chance of being completely placed into the main memory. It would also be interesting to study the impact of threshold detection heuristics when applied to other practical problems, such as document classification [Cristo et al. 2003].

REFERENCES

Baeza-Yates, R. and Ribeiro-Neto, B. *Modern Information Retrieval: the concepts and technology behind search.* Addison-Wesley, 2011.

Broder, A. Z., Carmel, D., Herscovici, M., Soffer, A., and Zien, J. Efficient Query Evaluation Using a Two-level Retrieval Process. In *Proceedings of the International Conference on Information and Knowledge Management.* New Orleans, USA, pp. 426–434, 2003.

Clarke, C., Craswell, N., and Soboroff, I. Overview of the TREC 2004 Terabyte Track. In *Proceedings of the Text Retrieval Conference.* Gaithersburg, USA, pp. 1–9, 2004.

Cristo, M., Calado, P., de Moura, E. S., Ziviani, N., and Ribeiro-Neto, B. A. Link Information as a Similarity Measure in Web Classification. In M. A. Nascimento, E. S. de Moura, and A. L. Oliveira (Eds.), *String Processing and Information Retrieval.* Lecture Notes in Computer Science, vol. 2857. Springer, pp. 43–55, 2003.

Ding, S. and Suel, T. Faster Top-k Document Retrieval Using Block-max Indexes. In *Proceedings of the International ACM SIGIR Conference on Research & Development of Information Retrieval.* Beijing, China, pp. 993–1002, 2011.

Robertson, S. E. and Walker, S. Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval. In *Proceedings of the International ACM SIGIR Conference on Research & Development of Information Retrieval.* Dublin, Ireland, pp. 232–241, 1994.

Rossi, C., de Moura, E. S., Carvalho, A. L., and da Silva, A. S. Fast Document-at-a-time Query Processing Using Two-tier Indexes. In *Proceedings of the International ACM SIGIR Conference on Research & Development of Information Retrieval.* Dublin, Ireland, pp. 183–192, 2013.

Salton, G., Wong, A., and Yang, C.-S. A Vector Space Model for Automatic Indexing. *Communications of the ACM* 18 (11): 613–620, 1975.

Strohman, T. and Croft, W. B. Efficient Document Retrieval in Main Memory. In *Proceedings of the International ACM SIGIR Conference on Research & Development of Information Retrieval.* Amsterdam, Netherlands, pp. 175–182, 2007.