An Incremental Approach to Efficient Pseudo-Relevance Feedback

Hao Wu
Department of Electrical and Computer Engineering
University of Delaware
Newark, DE USA
haow@udel.edu

Hui Fang
Department of Electrical and Computer Engineering
University of Delaware
Newark, DE USA
hfang@udel.edu

ABSTRACT

Pseudo-relevance feedback is an important strategy to improve search accuracy. It is often implemented as a two-round retrieval process: the first round is to retrieve an initial set of documents relevant to an original query, and the second round is to retrieve final retrieval results using the original query expanded with terms selected from the previously retrieved documents. This two-round retrieval process is clearly time consuming, which could arguably be one of main reasons that hinder the wide adaptation of the pseudo-relevance feedback methods in real-world IR systems.

In this paper, we study how to improve the efficiency of pseudo-relevance feedback methods. The basic idea is to reduce the time needed for the second round of retrieval by leveraging the query processing results of the first round. Specifically, instead of processing the expand query as a newly submitted query, we propose an incremental approach, which resumes the query processing results (i.e. document accumulators) for the first round of retrieval and process the second round of retrieval mainly as a step of adjusting the scores in the accumulators. Experimental results on TREC Terabyte collections show that the proposed incremental approach can improve the efficiency of pseudo-relevance feedback methods by a factor of two without sacrificing their effectiveness.

Categories and Subject Descriptors: H.3.3 [Information Search and Retrieval]: Search process, Relevance feedback

General Terms: Performance; Experimentation

Keywords: Pseudo-relevance Feedback; Incremental Approach; Efficiency; Query Expansion

1. INTRODUCTION

Pseudo-relevance feedback is an important technique that can be used to improve the effectiveness of IR systems. Different pseudo-relevance feedback methods have been proposed and studied for various retrieval models [8, 19, 26–28, 37, 41], but they all boil down to the problem of expanding an original query with useful term selected from a certain number of pseudo-relevant documents, i.e., top-ranked documents from an initial retrieval run. In particular, the implementation of pseudo-relevance feedback methods always require two rounds of retrieval, with the first round retrieving an initial set of documents with respect to an original query and the second round retrieving documents based on an expanded query, which is generated by expanding the original query with relevant terms selected from the previously retrieved document set.

Although pseudo-relevance feedback methods can lead to large performance gain in terms of effectiveness, there is one major downside that limits their usability in real retrieval application: its low efficiency [6, 13, 18]. In particular, the second round of retrieval could significantly slow down the performance due to the time spent on selecting terms for expansion and on executing the expanded query that are often much longer than the original one [13, 18]. It is clear that such drastically increased execution cost limits the applicability of pseudo-relevance feedback methods in real IR applications.

Compared with continual efforts on improving the effectiveness and robustness of pseudo feedback methods in the past decades [8, 10, 11, 14, 15, 19–21, 21, 22, 24, 26–28, 32, 37, 38, 41], less attention has been paid to make these methods more efficient [6, 13, 18]. Billerbeck and Zobel [6] proposed a summary based method for efficient term selection from the feedback methods. Lavrenko and Allan [18] proposed a fast relevance model, and Cartright et. al. [13] studied approximation methods to reduce the storage and computational cost for the fast relevance model. These proposed methods were designed for either a specific feedback method [13, 18] or a bottleneck in the feedback implementation, i.e., how to efficiently select expansion terms from the feedback documents [6]. However, it remains unclear whether there is a general solution to address other bottlenecks for efficient pseudo-relevance feedback implementation.

In this paper, we propose a general solution that can improve the efficiency of pseudo-relevance feedback methods. The basic idea is to reduce the execution cost for the expanded query by using the the query processing status of the original query. Existing IR systems often implement the pseudo-feedback methods as a two-round retrieval, where the second round is processed *independently* to the

SIGIR'13, July 28–August 1, 2013, Dublin, Ireland.

first round. However, since the query used in the second round retrieval is an expanded version of the original query used in the first round, it would be beneficial to leverage the results of the first round of retrieval to reduce the query processing time for the second round. Although this is an intuitive idea, we are not aware of any previous work that successfully implemented the idea. Experimental results on TREC Terabyte collections show that the proposed strategy can improve the efficiency by a factor of 2 to 4 without sacrificing the effectiveness.

The remainder of this paper is organized as follows. We first discuss related work on pseudo-relevance feedback in Section 2, and then provide background knowledge about indexing and query processing in Section 3. We then describe the motivation and details of the proposed incremental approach in Section 4. We explain the experiment set up, present experiment results and summarize our findings in Section 5. Finally, we conclude and discuss the future work in Section 6.

2. RELATED WORK

2.1 Pseudo-Relevance Feedback Methods

Pseudo-relevance feedback is an effective technique for improving retrieval accuracy. The commonly used feedback method for vector space models is the Rocchio algorithm [27,28]. In classical probabilistic models, the feedback method is based on the Robertson/Sparck-Jones weighting [26]. Two representative feedback methods for the language modeling approaches are the relevance model [19] and the model-based feedback [41].

Despite the difference in selecting expansion terms (i.e., estimating relevance models in language modeling framework), all these methods share similar implementation strategies, i.e., *initial retrieval* to find documents relevant to an original query; *term selection* to identify useful expansion terms from the feedback documents; and second-round retrieval to return documents relevant to the expanded query. This commonality in implementation makes it possible to derive a general optimization strategy that can improve the efficiency of these pseudo-relevance feedback methods, which is the focus of our paper.

More recently, continuous efforts have been put on improving the effectiveness and robustness of the pseudo-feedback methods [8,10,11,14,15,20–22,24,32,38]. For example, Tao and Zhai [32] extended model-based feedback methods using a regularized EM algorithm to reduce the parameter sensitivity, and Lv and Zhai [21] extended the relevance model to exploit term proximity.

2.2 Efficient Pseudo-Relevance Feedback

Compared with the effectiveness of an IR system, its efficiency is equally important. In particular, high query latency can decrease user satisfaction significantly [23]. Unfortunately, only a few previous studies focused on efficient pseudo-relevance feedback methods [5,6,13,18].

Billerbeck and Zobel [5,6] examined the bottlenecks of the pseudo-relevance feedback implementation and proposed to leverage tf-idf document summaries to reduce the cost of selecting expansion terms from the feedback documents. They also explored other approaches such as query association, using reduced-size collections and carrying accumulator information from the first round of retrieval to the second round, but all of them were unsuccessful. Our approach is in the same line of the third unsuccessful strategy they mentioned in the paper. However, we reach a different conclusion with our implementation, i.e., our method can improve the efficiency without obvious sacrificing the effectiveness. The main reason behind this fact is that our method allows the expanded terms to incrementally evaluate the accumulators in the initial retrieval rather than merely re-rank the top documents of the initial runs.

Lavrenko and Allan [18] proposed a fast relevance model which computes the original relevance-based ranking based on the cross-entropy between two documents (i.e., one representing the relevance model and the other representing the document to be scored). The re-arrangement used in the derivation makes it possible to shift the main computational cost from the query processing time to indexing time. Their experiment results show that the new method can reduce the query execution time significantly. However, the downside is the time taken to compute the document similarity matrix could be rather long (e.g., 90 hours for a small collection), which could makes it impractical for larger collections. Cartright et. al. [13] took one step further and propose approximation methods to reduce the storage and computational cost of this offline computing. Since these methods are specifically designed for relevance models, it remains unclear whether similar strategy can be leveraged for other feedback methods.

Cartright and Allan [12] have focused on optimizing the efficiency for interpolating subqueries. In particular, they investigated four cases of such an interpolation with varying amounts of accessibility for the original queries and expansion terms, and then proposed methods for each case. Our problem formulation is similar to the case when having access to both original queries and expansion terms, and our proposed approach bears similarity to the proposed Rewind algorithm. However, we use a different query processing technique, i.e,. score at a time (SAAT), from the one used in the previous study, i.e., document at a time (DAAT). Since SAAT makes it easier to resume the ranking status and adjust the document scores based on new terms, the improvement of our results is more significant than observed in the previous study.

Pseudo-relevance feedback is a specific type of query expansion methods where expansion terms are selected from highly-ranking documents. We now describe a couple of efforts on efficient query expansion [33, 35]. Theobald et. al. [33] proposed an optimization strategy for query expansion methods that are based on term similarities such as those computed based on WordNet. Their pruning method was designed based on a similarity based scoring function, and it remains unclear how to generalize it to other query expansion methods. Wang et. al. [35] proposed a solution for efficient query expansion for advertisement search. They focused on solving a domain specific challenge, i.e., the placement of similar bid phrases. The proposed strategies in these studies are very problem-specific, and can not be directly applied to pseudo-relevance feedback methods.

3. BACKGROUND ON INDEXING AND QUERY PROCESSING

Most current IR systems are based on inverted indexes [42]. An inverted index contains one inverted list for each term in the document collection. An inverted list of a term

contains a list of postings, and each posting describes the information of a document containing the term such as document ID and the term occurrence in the document. Query processing involves traversing the inverted lists of query terms, computing the relevance score for each document based on a retrieval function and then ranking the documents based on their scores.

Significant efforts have been made on making this process more efficient over the past decade [1–4,7,9,17,25,29,30,42]. Due to the limited space, we mainly provide some background about a few influential studies including those used in our system [1–4, 25, 29, 30].

3.1 Index Organization

Traditional inverted indexes are *document sorted indexing*, where the postings of an inverted list are sorted based on the document IDs [36]. The inverted lists of common terms could consist a huge number of postings. To enable faster access to the lists, various compression techniques such as delta encoding for document IDs have been used [42]. As the document IDs are usually sorted based on an increasing order, the delta codes are usually much smaller than the original IDs. It is therefore possible to reduce the space usage by storing delta codes instead of the original document IDs. Indexing compression not only saves the usage of disk and memory, but also improves the efficiency by minimizing cache/memory misses.

Another commonly used indexing organization strategy is *impact-sorted indexing*, where the postings are sorted based on their impact, i.e., their contribution to the relevance score of a document [1–4, 29]. With the impact-sorted indexing, the retrieval score of a document D for query Q can be evaluated by summing up the impact scores of D in the inverted lists of all the terms in Q. To further improve the efficiency, the impact scores are binned into a smaller number of distinct values and only the binned integer values are stored in the index. This strategy brings two benefits. First, the impact score can be stored using a very small number of bits (e.g., 6 bits for 64 distinct binned integers) instead of using 32 or 64 bits for a floating point variable. Second, with the binned scores, the number of documents with the same score is pretty large so that we can separate an inverted list into segments, one for each distinct value of the impact scores. Within each segment, documents are then ranked based on their document IDs, and the compression techniques used for document-sorted indexing can be applied here to achieve high level of compression. Formally, the retrieval score of document D for query Q is computed as follows,

$$S_{Q,D} = \sum_{w \in Q} B_{w,D}, \tag{1}$$

where $B_{w,D}$ is the binned integer score representing the impact of query term w in D. Previous studies have shown that such binning strategy would not significantly affect the retrieval accuracy [3, 29].

Skipping is another mechanism for fast accessing the information from the postings. Skips are forward pointers within an inverted list, and make it possible to pass over non-promising information on the postings with minimal efforts. They are often inserted into the indexes and stored as additional information. With the help of the skips, the system can jump to required records without going through the posting list one record by one record.

3.2 Query Processing

When processing a query, there are three different techniques to traverse the indexes and compute the relevance score for each document.

- *Document-At-A-Time (DAAT)* [7, 31, 34]: It assigns a document its final score before the next document is considered. In particular, the inverted lists of all query terms are processed in parallel. The iterators of the inverted lists need to frequently compare with each other to make sure they keep the same pace. Once all the postings of a document are accessed and processed, the system can forward the iterators to deal with the next document.

- *Term-At-A-Time (TAAT)* [9,25,42]: It starts with the inverted list of a query term, and would finish evaluating all the documents in the list before it moves to the next query term. As a result, an accumulator is required for each document to store and accumulate the retrieval scores. Once lists of all the query terms are processed, the documents with the K largest accumulated scores are returned as the retrieval results.

- *Score-At-A-Time (SAAT)* [4, 29, 30]: This strategy is only suitable for impact-based indexing. It fetches all inverted lists first (as in DAAT) and processes the postings in the decreasing order of the impact values instead of document IDs as used in TAAT. Document accumulators are still needed to accumulate partial score contributions from the different inverted lists. Since SAAT places the most promising postings close to start of the lists, early termination techniques are often used to improve the efficiency [4, 16, 39].

To reduce the computational efforts, pruning has been introduced to all these three strategies [4, 9, 34] with the goal of limiting the number of accumulators and the scan depth on the index lists. The main idea is that the process can be terminated once the system identifies that further processing will not change the ranking of top k documents.

3.3 What We Use in This Paper

We follow the previous studies [4, 29] and use a state-of-the-art four-stage SAAT query processing strategy with the impact-based indexing since it can achieve very good performance in many cases. Note that we use query generation model with Dirichlet prior smoothing as the basic retrieval model [40] (i.e., the one without using feedback) and the relevance model [19] as the pseudo-relevance feedback method. However, our proposed incremental approach can be applied to other retrieval functions and feedback methods and we plan to study it as one of our future work.

To compute the impact scores (i.e., $B_{w,D}$ in Equation (1), we use the derived equation from previous study [29]:

$$B_{w,D} = \lfloor \frac{n}{M + log(s)} (logP(w|D) - C_w) \rfloor,$$

where n is the number of distinct values used to represent impact scores and set to 64. C_w is the minimum value of $P(w|D)$ in the collection (i.e., $C_w = \min_D logP(w|D)$). M is used to ensure that the binned score is within the range, and s is a saturation parameter which makes sure that all the bin values, i.e., $[0, n]$, can be fully utilized. The details

of the derivation can be found in the references [29]. Moreover, the term probability is estimated using Dirichlet Prior smoothing [40], i.e.,

$$P(w|D) = \frac{c(w;D) + \mu c(w;C)/|C|}{|D| + \mu}.$$

The parameter μ is set to 2,500 and the skip length is set to 128 bytes.

For query processing, we use the SAAT with a four-stage pruning. The four stages are: OR, AND, REFINE and IGNORE. The processing begins with the OR stage by processing the postings in the inverted lists based on the decreasing order of the impact values. Document accumulators are created whenever it is necessary, i.e., when a new document is seen in one of the inverted lists. The processing switches to AND stage when we can prove that we have created accumulators for all the documents that could possibly enter the top n. In AND stage, we ignore all the documents without an accumulator. The processing continues and we update the scores for the existing accumulators with newly processed information. The processing switches to REFINE stage as soon as we know exactly what the top n documents are without knowing their exact ranking. REFINE stage works with the accumulators of the top n documents. Once we can determine that rank order of the top n documents, the process enters the final IGNORE stage, which means that all the remaining information from the inverted list can be ignored.

The basic idea of the pruning strategy is to gradually reduce the number of active accumulators when we gain more confidence on knowing that other accumulators would not change the final top-k search results. In particular, OR stage is the only one that can add accumulators, AND stage only updates existing accumulators, and REFINE stage only process accumulators that can make to the top k results. Moreover, we further speed up the process by using skipping and accumulator trimming [29]. Skipping enables fast processing of long postings, while accumulator trimming can reduce the computational cost in the AND stage by dynamically reduce the number of enabled accumulators.

4. EFFICIENT PSEUDO-RELEVANCE FEEDBACK

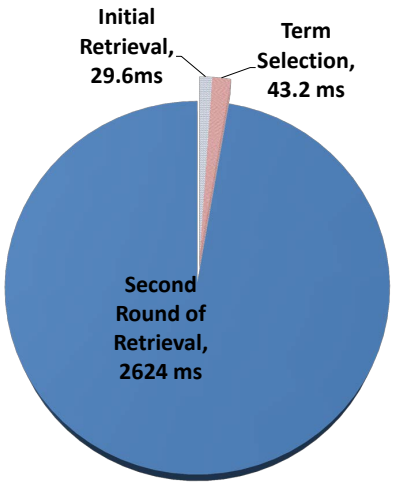
4.1 Overview of Existing Implementation Strategy

As described in Section 2.1, many pseudo-relevance feedback methods have been proposed and studied. Despite the differences on how to exploit feedback information, they all require the following three-step implementation:

1. **Initial retrieval:** finding documents relevant to an original query;
2. **Term selection:** identifying useful expansion terms (i.e., relevant information in language modeling framework) from the feedback documents;
3. **Second-round retrieval:** returning documents relevant to the expanded query (i.e., updated query model in the language modeling framework).

We now provide more details on how each of these three steps is implemented in existing IR systems.

Figure 1: Processing time of the three steps in pseudo-relevance feedback implementation



The *initial retrieval* step can be implemented with any existing top-k query processing techniques as described in Section 3. There have been significant efforts on optimizing the efficiency for this step [1–4, 7, 9, 17, 25, 29, 30, 42].

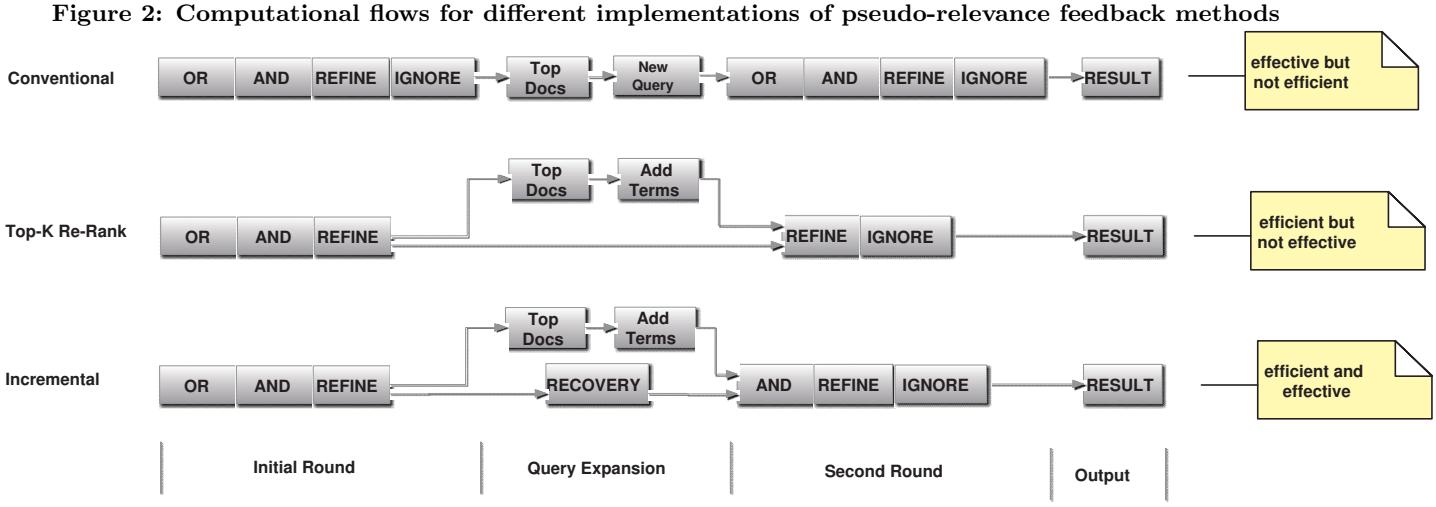
The *term selection* step is to select important terms from the feedback documents with the expectation that the selected terms can bring more relevant documents in the second round of retrieval. Traditionally, these terms are selected directly from the top ranked documents. However, this step could takes lots of time when the documents are long and the information about the documents need to be read from the disk. To solve this problem, Billerbeck and Zobel [6] proposed to generate a short tf-idf based summary for each document and select expansion terms from the summaries of the feedback documents. These summaries are small enough to be pre-loaded into the memory, and can lead to more efficient term selection. Their experimental results showed that this strategy is efficient with ignorable loss in terms of the effectiveness. In this paper, we use this strategy for term selection. One difference is that we use term probability $p(t|D)$ instead of tf-idf weighting to generate document summaries because the retrieval function used is based on language modeling approach. And we set the length of document summary to 20 terms.

The *second round retrieval* step aims to retrieve final retrieval results with the expanded query. The expanded query is often formulated as a linear interpolation of the original query and the expansion terms selected from the second step. As an example, in the relevance model [19], this step is to retrieve documents with an updated query model (i.e., θ_Q^{new}) by linearly combining the original query model (i.e., θ_Q) with the relevance model estimated from the feedback documents (i.e., $\theta_{\mathcal{F}}$) as follows:

$$\theta_Q^{new} = \lambda \cdot \theta_Q + (1 - \lambda) \cdot \theta_{\mathcal{F}}, \quad (2)$$

where the original query model θ_Q is estimated using the maximum likelihood estimation of query Q, the relevance model $\theta_{\mathcal{F}}$ estimated from the feedback documents $\mathcal{F}$ using

Figure 2: Computational flows for different implementations of pseudo-relevance feedback methods



the methods described in previous study [19], and λ is to control the amount of feedback.

In the second round of retrieval, existing IR systems such as Indri would process the expanded query in the same way as a newly submitted query. In other words, the two rounds of retrieval are processed *independently*.

Figure 1 shows how much time each step takes when using the existing implementation methods described above for the relevance feedback method with 20 expansion terms. It is clear that the third step takes the most of computational time while the other two stages share a very small part the time usage. Thus, the key to efficient pseudo-relevance feedback methods is to reduce the execution time for the *second-round retrieval*, which is the focus of our paper.

4.2 Analyzing the second round retrieval

Compared with the initial retrieval, the expanded query processed in the second round is often much longer than the original query because it has much larger number of query terms to be processed. For example, the average length of Web queries is around 3, while the number of expansion terms is often set to 20. As a result, the computational cost for expanded query is significantly higher than that for the basic retrieval. Take the implementation used in our paper as an example, a longer query means that each accumulator needs to collect more postings to produce a final result and more accumulators would be created and evaluated. As a result, the system performs much more index accesses and computing, which leads to extremely long processing time compared with that for shorter queries.

One limitation of existing implementation for feedback methods is that the two rounds of retrieval are processed *independently*. Each round starts with an empty set of accumulators and gradually adds new accumulators in the OR stage. When switched to AND stage, accumulators can be updated but no new accumulated can be added. When switched to REFINE stage, only top k accumulators are processes. And in the final IGNORE stage, all the information from the inverted lists can be ignored. Note that the accumulators used the two rounds of retrieval are computed from the scratch separately. This implementation is illustrated in the upper part of Figure 2

However, unlike processing a new query, the expanded query in the second round retrieval is related to the query used in the initial retrieval. In particular, the query terms in the initial retrieval is a subset of those in the second round of retrieval, and these terms will be processed twice in this two rounds of retrieval process. Moreover, the results of these two rounds of retrieval might have a great overlap. Thus, it would be interesting to study how to leverage the results of the first round of retrieval to reduce the computational cost. But how to leverage them? This is not a simple problem without significant challenges.

The idea of exploiting the results of initial retrieval to improve the efficiency of the second round retrieval was discussed in previous study [6], but was not found useful. In the next subsection, we re-visit this basic idea and propose an incremental approach that is shown to be both efficient and effective based on the experimental results.

4.3 The Proposed Incremental Approach

Our basic idea is that the initial retrieval process should be treated as part of the query processing for the second round retrieval. Instead of processing the expanded query in the second round from the scratch, we should be able to resume the query processing results of the initial query, and continue the processing for the expanded query terms. But how to resume the results?

One possible strategy is to resume the last ranking-related stage, i.e., REFINE, in the query processing results of the initial retrieval. Recall that REFINE stage is designed to process only the accumulators of top K ranked documents. Thus, if we resume the status from the REFINE stage in the second round retrieval, it is equivalent to re-ranking those top K ranked documents using the expanded query. This strategy is illustrated in the middle part of Figure 2. Since the number of accumulators used in REFINE stage is very small, this re-ranking method would be quite efficient. However, it would suffer significant loss in terms of the effectiveness because one of the major benefits of feedback methods is to find relevant documents that were not among top ranked results for the initial retrieval and the re-ranking strategy seems to disable this nice benefit. Clearly, this is not an optimal solution. Can we do better?

Intuitively, if more document accumulators can be included in re-ranking process, the retrieval effectiveness could be improved. On the extreme case, when all the documents are considered for the re-ranking, the cost would be the same as submitting a new query. Thus, the main challenge here is how to select a set of documents to be re-ranked so that we can increase the efficiency without sacrificing the effectiveness.

Recall that the pruning technique consists of four stages: OR, AND, REFINE and IGNORE. The number of active accumulators becomes smaller as the system switches from one stage to the other. As discussed earlier, resuming from the REFINE stage is equivalent to re-ranking only top k documents, which hurts the effectiveness. On the contrary, resuming from the OR stage would not hurt the effectiveness. However, since the number of expanded terms is much larger than the number of original query terms, we might not be able to reduce the number of accumulators significantly. Thus, we propose to resume from the AND stage.

The main idea of our incremental approach is shown in the lower part in Figure 2. REFINE is the last ranking-related stage in the initial retrieval. Thus, in order to resume from the AND stage of the initial retrieval, we have to first rewind the query processing results from the end of REFINE stage back to the end of the AND stage. This new stage is referred to as RECOVERY stage in our system. The RECOVERY stage has two tasks: (1) re-enable the accumulators that were disabled at the REFINE stage; and (2) turn back the inverted list pointers to the positions where they were at the end of AND mode. Our experimental results show that this stage takes a very short ignorable time. After the RECOVERY stage, we will switch to the new AND stage to update the accumulators based on the expanded terms, and then continue to the other two stages as usual.

4.4 Discussions

Our proposed incremental approach can improve the efficiency because of the following two reasons.

First, query processing results of the original query terms can provide useful information for effective pruning in the second round retrieval. Specifically, accumulator trimming is used in the AND stage to dynamically reduce the number of active accumulators, and a threshold is used to decide whether an accumulator should be kept active or not. The threshold is to estimate a lower bound of the relevance scores for the top K ranked documents. This threshold is set to $-inf$ at the beginning of the retrieval process, and will be updated at the later stage of the retrieval when more information is gathered. Since initial value of the threshold is rather small, little pruning is applied at the early stage of the retrieval process. If the system could be informed with the range of the threshold, more pruning can be done, which would lead to shorter processing time. Since resuming the process of the initial retrieval can provide a much larger initial value for the pruning threshold, the proposed approach can reduce the query processing time.

Second, the efficiency is closely related to the number of accumulators that need to be processed. Long queries usually lead to a huge number of accumulators, which significantly hurt the efficiency. However, when resuming the query processing results of the initial query, we are able to start with a much smaller set of accumulators. Note that this strategy is not ranking safe when the expanded query is significantly different from the initial query (i.e., when λ in Equation 2 is very small). However, as shown in Section 5, the optimal value of λ is often large and this strategy does not affect the effectiveness significantly.

Table 1: Efficiency comparison using TREC Terabyte ad hoc queries (i.e., average query processing time (ms) to retrieve 1K documents)

Data sets	NoFB	FB-BL	FB-Incremental
TB04	105	5,522	3,023
TB05	77	4,502	2,462
TB06	71	4,542	2,082

Table 2: Comparison of the average size of accumulator lists per query

	FB-BL	FB-Incremental
TB04	2,368K	370K
TB05	1,805K	252K
TB06	1,901K	240K

5. EVALUATION

5.1 Experiment Design

In our study, we use three standard TREC data sets which were created for TREC Terabyte tracks in 2004 to 2006. These three sets are denoted as TB04, TB05 and TB06 in this paper. All the data sets use Gov2 collection as the document set, and the collection consists of 25.2 million web pages crawled from the .gov domain. All experiments were conducted on a single machine with dual AMD Lisbon Opteron 4122 2.2GHz processors, 8GB DDR3-1333 memory and four 2TB SATA2 disks.

The basic retrieval model used in our experiments is the Dirichlet Prior smoothing method [40], where the parameter μ was set empirically to 2500. This method is labeled as **NoFB**. We use the relevance model [19] as the pseudo-relevance feedback method. This model is chosen because of the following two reasons. First, it is a state-of-the-art pseudo-relevance feedback method that has been shown to be both effective and robust. Second, it is implemented in the Indri [1] toolkit, which makes it possible to verify the correctness of our implementation. This method is labeled as **FB**.

The implementation of our basic retrieval system (i.e., **NoFB**) is described in Section 3.3. Based on the basic retrieval system, we implemented two pseudo-relevance feedback systems: (1) **FB-BL:** traditional method of implementing pseudo-relevance feedback methods, i.e., the expanded query is processed independently to the original query; (2) **FB-Incremental:** our proposed approach that exploits the query processing results of the original query to speed up the processing of the expanded query.

Although we implemented our own indexing and query processing modules, we did not build the impact-based indexing directly from the collection. Instead, we built an initial index using Indri toolkit and use Indri API to transfer the indri index to our impact-sorted index. The size of the impact-based index for Gov2 collection is about 11.8GB with 442MB for term lookup, 7.5GB for inverted lists and

[1]http://www.lemurproject.org/indri/

Table 4: Impact of the expansion weight (i.e., λ) on the efficiency (the average execution time (ms) per query)

Data sets	Methods	λ				
		0.5	0.6	0.7	0.8	0.9
TB04	FB-BL	6,125	5,522	4,435	2,723	1,597
	FB-Incremental	3,798	3,023	2,130	1,221	826
TB05	FB-BL	5,146	4,502	3,577	2,466	1,323
	FB-Incremental	3,071	2,462	1,762	1,113	845
TB06	FB-BL	5,281	4,542	3,571	2,412	1,114
	FB-Incremental	2,664	2,082	1,406	778	560

Table 3: Effectiveness comparison using TREC Terabyte ad hoc queries (MAP@1000)

Data sets	NoFB	FB-BL	FB-Incremental
TB04	0.246	0.256	0.255
TB05	0.309	0.334	0.334
TB06	0.275	0.285	0.284

3.8GB for document summaries that are needed for the term selection step in the feedback methods. We decided to leverage the Indri index because (1) it saves our unnecessary efforts on writing codes to parse documents and count the term statistics; and (2) it enables direct comparison between our system and Indri since they now use the same information about the collection. When evaluating a query, we also leveraged Indri API to handle the tasks of converting terms/documents to their IDs.

Since we have to work with two indexes (the original Indri index and new Impact-sorted index) and all the information are read directly from disk rather than memory, our system could be slower than those using similar strategies. However, this should not affect our comparison on the two FB systems since they are implemented using the same strategy. Moreover, even working with two indexes, our developed system is still much faster than the Indri toolkit (version 5.1), which takes about 1 minute to process a query when using the relevance model on the Gov2 collection.

5.2 Experimental Results

The first two sets of experiments were conducted with the ad hoc queries used in TREC Terabyte tracks. Each data set has 50 queries, and we use title only field to formulate queries. The third set of experiments was conducted with the efficiency queries used in TREC Terabyte tracks. One data set has 50K queries while the other has 100K queries.

5.2.1 Efficiency and Effectiveness

We first examine whether the incremental approach can improve the efficiency. In particular, we set the number of feedback documents to 10, the number of expansion terms to 20 and the expansion weight λ is set to 0.6. These parameters are chosen because they can lead to near-optimal performance in terms of the effectiveness. Table 1 shows the average processing time (ms) for each query when 1000 documents are retrieved. The trends on the three collections are similar. Both FB systems are slower than the NoFB system. It takes around 5 seconds for FB-BL to process a query, and takes around 2.5 seconds for FB-Incremental to do so. Clearly, FB-Incremental can achieve a speed up of 2 compared with FB-BL, which shows that the proposed incremental approach can improve the efficiency. Note that we also evaluate the efficiency of the Indri toolkit, a widely used open source IR system. It takes around 1 minute for the Indri toolkit to process a query on the same collection. This suggests that the baseline system we implemented, i.e., FB-BL, is very efficient.

As discussed earlier, the speed up of FB-Incremental is achieved for two reasons. First, FB-Incremental leverages the query processing results of the initial retrieval, which avoids processing the original query terms twice. Second, the accumulator list inherited from the initial retrieval by FB-Incremental is much smaller. Table 2 shows the comparison of the accumulator list size between the two methods. Our incremental method creates only about 1/7 accumulators as those of baseline method. As a result, the new method avoids a lot of unnecessary index access and computing for those accumulators it does not include. Note that our baseline system, i.e., FB-BL, is a very strong baseline since it applied the accumulator trimming techniques which can reduce unnecessary accumulators during the process.

We also conduct experiments to examine whether the proposed approach would hurt the performance. Since our incremental approach leverages the accumulators used in the AND stage of the initial retrieval, it is possible that we may miss some relevant documents because they were not relevant to the original query and thus were not included in its accumulator lists. As a result, it is possible that the incremental approach might hurt the effectiveness, but it only happens when the expanded query is very different from the original query. Recall that the optimal value of λ is 0.6 which indeed indicates that the original query and expanded query are similar. Table 3 shows the results measured with MAP@1000. To ensure the correct implementation of our system, we compare the baseline performance with the ones reported in the previous study [30] and find that the results are similar, which confirms our implementation of $NoFB$ is correct. Moreover, we find that both FB methods can improve the retrieval accuracy about 4% to 8%, which is also similar to the performance improvement of the feedback method implemented in Indri. One interesting observation is that the effectiveness of the two FB systems are similar, which indicates that our proposed approach improve the efficiency without sacrificing the effectiveness.

5.2.2 Impact of Parameter Values

There are multiple parameters in the pseudo-relevance feedback methods such as the number of expansion terms, the number of feedback documents and the expansion weight (i.e., λ in Equation (2)). Since we use impact based document summary for term select, the impact of the number of feedback documents on efficiency is not very significant. Thus, we only focus on the other two parameters.

Table 5: Impact of the expansion weight (i.e., λ) on the effectiveness (MAP@1000)

Data sets	Methods	λ				
		0.5	0.6	0.7	0.8	0.9
TB04	FB-BL	0.254	0.256	0.257	0.255	0.252
	FB-Incremental	0.252	0.255	0.256	0.255	0.252
TB05	FB-BL	0.333	0.334	0.332	0.328	0.321
	FB-Incremental	0.333	0.334	0.332	0.327	0.321
TB06	FB-BL	0.284	0.285	0.285	0.284	0.281
	FB-incremental	0.282	0.284	0.283	0.282	0.280

Table 6: Average query execute time (ms) based on different expansion weights (top 10 documents for each query)

	NoFB	FB Methods	λ				
			0.5	0.6	0.7	0.8	0.9
TB04	69	FB-BL	4,731	4,016	2,786	1,756	933
		FB-Incremental	1,295	915	578	437	403
TB05	53	FB-BL	3730	2,923	1,999	1,131	547
		FB-Incremental	983	709	451	341	310
TB06	46	FB-BL	3909	3,060	2,167	1,317	573
		FB-Incremental	921	669	447	319	284

Figure 3: The speed-up rate on different expansion weight λ (top 1000 documents returned)

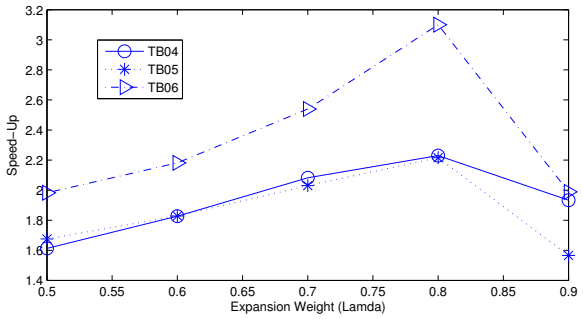
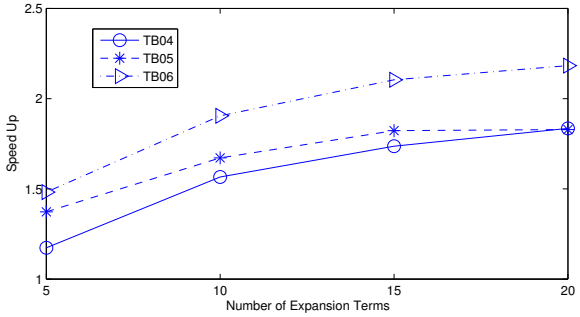


Figure 4: The speed-up rate on different number of expansion terms



As discussed earlier, the expansion weight is an important parameter and we should examine the impact of its value on both efficiency and effectiveness. The results are shown in Table 4 and Table 5. We see that the optimal value of λ is around 0.6, which means that we should put more weights to the original query terms. According to Equation (2), the higher value of λ means that we put more trust on the original query and less weights to the expansion terms. Given the characteristics of SAAT pruning technique, postings of terms with smaller weights are more likely to be pruned. As a result, we can observe a clear trend that both methods can improve the efficiency more when the value λ is larger.

Figure 3 shows how the speed-up rate of $FB{-}Incremental$ compared with FB changes with the expansion weight. The higher expansion weight we choose, the more the original query determines the final ranking list and the more time the incremental method can save. However when the expansion weight is too high (e.g. 0.9), SAAT pruning technique is rather efficient and leave small room for further improvement. As a result, the speed-up rate at high expansion weight decreases.

We further test the impact of number of expansion terms on efficiency. Table 4 shows the speed up rate of $FB - Incremental$ compared with $FB - BL$. It shows that our new method benefits more when the system adds more expansion terms into the original query. The main reason of this trend is that our method efficiently controls the grows of accumulator list size when the query becomes longer.

Another important factor that could affect the retrieval speed is the number of documents retrieved for each query. Figure 5 shows how the number of retrieved documents affect the speed up of $FB - Incremental$ over $FB - BL$. The results indicate that the new method can achieve more speed-up when the system returns fewer results to users. We believe it is due to the reason that returning fewer documents brings higher cut-off threshold. And in our method, the high cut-off threshold which is got from the initial run provides a strong and efficient guidance on the accumulator selections in the resumed retrieval. As a result, the pool of candidate documents/accumulators is kept in a very small size and final result is generated soon. In opposite, the baseline method which does the second round retrieval independently cannot get the benefit from the high cut-off threshold and it still generates large number of accumulators in which most of them are unnecessary.

Figure 5: The speed-up rate on different number of retrieved documents per query

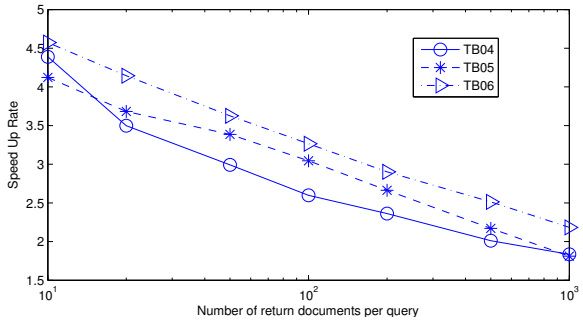
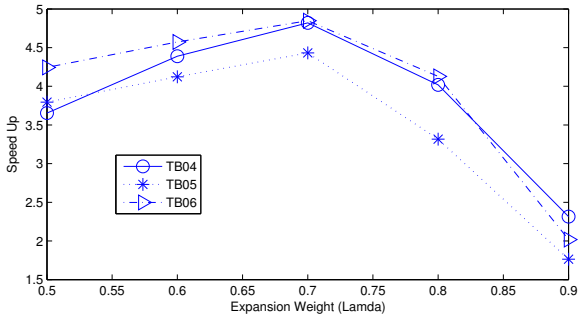


Figure 6: The speed-up rate on different expansion weight λ (top 10 documents returned)



We also report the efficiency when retrieving only 10 documents for each query in Table 6. It is clear that when retrieving fewer documents, FB-Incremental can achieve a higher speed up, i.e., around four. This indicates that this method could be very useful for Web search domain since search users only need to look at 10 results per page. Finally, Figure 6 shows impact of the expansion weight on the speed-up when only 10 documents are retrieved. The trend is very similar to Figure 3.

5.2.3 *Scalability*

To further test the scalability of the incremental method and study how original query length affects the pseudo feedback system speed, we test both methods on 50k 2005 efficiency queries and 100k 2006 efficiency queries. For each query, 5 expansion terms are added and it only returns top 10 documents. The results show that the incremental method can stably improve the efficiency at a rate more than 2 at different original query length. For short queries, the baseline is fast and it leaves less space for further improvement. For long queries, the processing time is dominated by the original query terms and the optimizations in pseudo feedback part will not affect the total time too much. As a result, it is reasonable to observe a speed up rate summit at medium length query (i.e., 3-4 terms).

Table 7: Average query processing time (ms) on different query length (2005 50K queries)

	Avg	1	2	3	4	5	>5
NoFB	35	4	10	26	55	91	175
FB-BL	297	38	94	302	526	749	1,102
FB-Incremental	124	18	43	97	204	316	540
Speed-up	2.4	2.1	2.2	3.1	2.6	2.4	2.0

Table 8: Average query processing time (ms) on different query length (2006 100K queries)

	Avg	1	2	3	4	5	> 5
NoFB	103	3	12	39	81	132	275
FB-BL	739	39	99	357	690	1,030	1,642
FB-Incremental	300	26	43	110	248	441	797
Speed-up	2.5	1.5	2.3	3.2	2.8	2.3	2.0

6. CONCLUSIONS

This paper focuses on efficient implementation of pseudo-relevance feedback methods, an important yet under-studied problem. Motivated by the fact that original query terms are often processed twice in the two-round retrieval process, we proposed an incremental approach that can exploits the query processing results of the first round retrieval for computing the document scores in the second round. Although similar idea was mentioned in a previous study [6], it was concluded as a unsuccessful strategy. On the contrary, our method has been shown to be useful and can achieve a speed up of 2 over TREC collections. This paper is one of a few studies that try to bring the gap between the continuous research efforts on improving the effectiveness of the pseudo-relevance feedback methods and the increasing efforts on efficient IR systems.

There could be many interesting directions for our future work. First, we plan to implement other pseudo feedback methods including both basic models such as model-based feedback [41] and more sophisticated models such as positional relevance model [22], and evaluate whether the proposed approach can improve their performance. Second, it would be interesting to study how to leverage the incremental approach to improve the efficiency of query processing for long queries. Finally, our system is built on impact-sorted indexing with SAAT traverse. We plan to continue our efforts and study whether it is possible to improve the efficiency for feedback methods when using other query processing strategies such as DAAT and TAAT .

7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Number IIS-1017026. We thank the anonymous SIGIR reviewers for their useful comments.

8. REFERENCES

[1] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proceedings of SIGIR'01*, 2001.

[2] V. N. Anh and A. Moffat. Impact transformation: efffective and efficient web retrieval. In *Proceedings of SIGIR'02*, 2002.

[3] V. N. Anh and A. Moffat. Simplified similarity scoring using term ranks. In *Proceedings of SIGIR'05*, 2005.

[4] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *Proceedings of SIGIR'06*, 2006.

[5] B. Billerbeck and J. Zobel. Techniques for efficient query expansion. In *Proceedings of String Processing and Information Retrieval Symposium*, 2004.

[6] B. Billerbeck and J. Zobel. Efficient query expansion with auxiliary data structures. *Information Systems*, 31(7):573–584, 2006.

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

[8] C. Buckley. Automatic query expansion using SMART: Trec-3. In D. Harman, editor, *Overview of the Third Text Retrieval Conference (TREC-3)*, pages 69–80, 1995. NIST Special Publication 500-225.

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

[10] C. Buckley and S. Robertson. Relevance feedback track overview: Trec 2008. In *Proceedings of TREC'08*, 2008.

[11] G. Cao, J.-Y. Nie, J. Gao, and S. Robertson. Selecting good expansion terms for pseudo-relevance feedback. In *Proceedings of SIGIR'08*, 2008.

[12] M.-A. Cartright and J. Allan. Efficiency optimizations for interpolating subqueries. In *Proceedings of the CIKM'11*, 2011.

[13] M.-A. Cartright, J. Allan, V. Lavrenko, and A. McGregor. Fast query expansion using approximations of relevance models. In *Proceedings of the CIKM'10*, 2010.

[14] K. Collins-Thompson and J. Callan. Estimation and use of uncertainty in pseudo-relevance feedback. In *Proceedings of SIGIR'07*, 2007.

[15] J. V. Dillon and K. Collins-Thompson. A unified optimization framework for robust pseudo-relevance feedback algorithms. In *Proceedings of CIKM'10*, 2010.

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

[17] R. Fagin. Combining fuzzy information: an overview. *ACM SIGMOD Record*, 31(2), 2002.

[18] V. Lavrenko and J. Allan. Real-time query expansion in relevance models. Technical Report IR-473, University of Massachusetts Amherst, 2006.

[19] V. Lavrenko and B. Croft. Relevance-based language models. In *Proceedings of SIGIR'01*, pages 120–127, Sept 2001.

[20] K.-S. Lee, W. B. Croft, and J. Allan. A cluster-based resampling method for pseudo-relevance feedback. In *Proceedings of SIGIR'08*, 2008.

[21] Y. Lv and C. Zhai. Positional relevance model for pseudo-relevance feedback. In *Proceedings of SIGIR'10*, 2010.

[22] Y. Lv, C. Zhai, and W. Chen. A boosting approach to improving pseudo-relevance feedback. In *Proceedings of SIGIR'11*, 2011.

[23] M. Mayer. Scaling google for every user. In *Seattle Conference on Scalability*, 2007.

[24] J. Miao, J. Huang, and Z. Ye. Proximity-based rocchio's model for pseudo relevance. In *Proceedings of SIGIR'12*, 2012.

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

[26] S. Robertson and K. Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27:129–146, 1976.

[27] J. Rocchio. Relevance feedback in information retrieval. In *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313–323. Prentice-Hall Inc., 1971.

[28] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 44(4):288–297, 1990.

[29] T. Strohman. *Efficient processing of complex features for information retrieval*. PhD thesis, University of Massachusetts Amherst, 2007.

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

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

[32] T. Tao and C. Zhai. Regularized estimation of mixture models for robust pseudo-relevance feedback. In *Proceedings of SIGIR'06*, 2006.

[33] M. Theobald, R. Schenkel, and G. Weikum. Efficient and self-tuning incremental query expansion for top-k query processing. In *Proceedings of the SIGIR'05*, 2005.

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

[35] H. Wang, Y. Liang, L. Fu, G. rong Xue, and Y. Yu. Efficient query expansion for advertisement search. In *Proceedings of SIGIR'09*, 2009.

[36] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.

[37] J. Xu and W. B. Croft. Improving the effectivness of information retrieval with local context analysis. *ACM Transactions on Information Systems*, 18:79–112, 2000.

[38] Y. Xu, G. J. F. Jones, and B. Wang. Query dependent pseudo-relevance feedback based on wikipedia. In *Proceedings of SIGIR'09*, 2009.

[39] H. Yan, S. Shi, F. Zhang, T. Suel, and J.-R. Wen. Efficient term proximity search with term-pair indexes. In *Proceedings of CIKM'10*, 2010.

[40] C. Zhai and J. Lafferty. The dual role of smoothing in the language modeling approach. In *Proceedings of the Workshop on Language Modeling and Information Retrieval*, 2001.

[41] C. Zhai and J. Lafferty. Model-based feedback in the KL-divergence retrieval model. In *Tenth International Conference on Information and Knowledge Management (CIKM 2001)*, pages 403–410, 2001.

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