

# Durable Top-k Search in Document Archives

Leong Hou U<sup>†</sup>, Nikos Mamoulis<sup>†</sup>, Klaus Berberich<sup>‡</sup>, Srikanta Bedathur<sup>‡</sup>

<sup>†</sup>Department of Computer Science, University of Hong Kong

Pokfulam Road, Hong Kong

{hleongu, nikos}@cs.hku.hk

<sup>‡</sup>Max-Planck Institute for Informatics

Saarbrücken, Germany

{kberberi, bedathur}@mpi-inf.mpg.de

## ABSTRACT

We propose and study a new ranking problem in versioned databases. Consider a database of versioned objects which have different valid instances along a history (e.g., documents in a web archive). Durable top- $k$  search finds the set of objects that are consistently in the top- $k$  results of a query (e.g., a keyword query) throughout a given time interval (e.g., from June 2008 to May 2009). Existing work on temporal top- $k$  queries mainly focuses on finding the most representative top- $k$  elements within a time interval. Such methods are not readily applicable to durable top- $k$  queries. To address this need, we propose two techniques that compute the durable top- $k$  result. The first is adapted from the classic top- $k$  rank aggregation algorithm NRA. The second technique is based on a shared execution paradigm and is more efficient than the first approach. In addition, we propose a special indexing technique for archived data. The index, coupled with a space partitioning technique, improves performance even further. We use data from Wikipedia and the Internet Archive to demonstrate the efficiency and effectiveness of our solutions.

## Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: Search process

## General Terms

Algorithms, Experimentation

## Keywords

Document Archives, Top- $k$  Search, Temporal Queries

## 1. INTRODUCTION

Consider a set of objects (e.g., web documents) and a sequence of different rankings of these objects. The rankings are ad-hoc (i.e., not pre-defined) and could be derived from a search operation (e.g., a keyword query). Assume that the objects are not static, but change over time (e.g., different versions of web pages), and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

Copyright 2010 ACM 978-1-4503-0032-2/10/06 ...\$10.00.

that the search operation refers to a time interval. Then, the different rankings are sensitive to the change of documents during the query interval. The Internet Archive ([www.archive.org](http://www.archive.org)) is a characteristic example of a document archive, where search on different versions of documents is possible. A given time interval (e.g., June 2008 – October 2009) and a set of keywords (e.g., “Welsh football player”) define a sequence of rankings of all documents over time. The order of a document may change if the document is replaced by a newer version that has different relevance to the keywords.

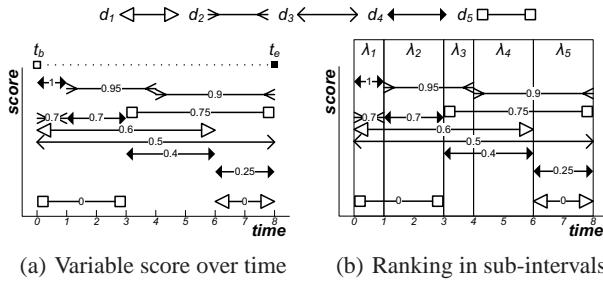
This paper studies the problem of finding objects that are consistently in the top- $k$  throughout the sequence of rankings defined by a time interval  $[t_b, t_e]$  and a set of keywords  $W$ . The main application is finding documents that are consistently relevant to a specific subject over a given time period. The result of this query has size 0 to  $k$ ; queries can have empty results if  $k$  is small or the rankings change radically. Empty results can be avoided by relaxing the consistent top- $k$  constraint of the query using a ratio variable  $r$ ,  $0 < r \leq 1$ : we seek for objects that are in the top- $k$  for at least  $r \times (t_e - t_b)$  time in the  $[t_b, t_e]$  interval. We call this problem *durable top- $k$  search*. Wikipedia is one of the systems where durable top- $k$  search can be applied. A page in Wikipedia is typically modified by editors over time. For instance, Figure 1 shows how an entry about football player “Ryan Giggs” evolves; this page has been modified over 3500 times from 2004 to present.



Figure 1: Different versions of topic *Ryan Giggs* in Wikipedia

As an example, consider five documents  $d_1-d_5$ , having different versions over time, and a query defined by time interval  $[t_b, t_e]$  and a set of keywords. The score (relevance) of each document over time, within the interval  $[t_b, t_e]$ , normalized to be within  $[0, 1]$  is shown in Figure 2(a). For instance, document  $d_4$  has four versions, with scores 1, 0.7, 0.4, and 0.25. Figure 2(b) shows the sub-intervals of  $[t_b, t_e]$ , within which the ranking remains constant. A crisp durable top-3 query with  $r=1$ , has  $d_2$  as the only result. If the query is relaxed to  $r = 0.6$ ,  $\{d_1, d_2, d_5\}$  becomes the query result.

This query is not only applicable to document archives, but in general for applications that need to merge of ad-hoc rankings, which are time-parametric. For example, consider the changing attributes of stocks over time (e.g., price, volume, etc.) and a consistent top- $k$  query for an aggregate (e.g., average) of an ad-hoc subset of these attributes. Other applications include expert finding



**Figure 2: Relevance of documents over time**

and finding information sources that one should subscribe to. For the former, consider a publication database and the query ‘column stores’. Instead of the documents, the authors are ranked and their aggregate score is derived from the scores of their relevant publications. Our query finds people who have consistently produced relevant work: these are considered long-time experts on the topic. For the latter, consider information sources such as blogs (twitter users) that one typically subscribes to (follows). The user may want to subscribe to those that consistently include relevant material to a set of keywords.

Berberich et al. [6] introduced time-travel keyword queries in document archives. Given a time interval and an aggregate function (relevance model), a time-travel keyword query returns the most relevant documents to the keywords according to their aggregate scores computed over the time interval. Typical relevance models compute the maximum (MAX), minimum (MIN), or average (AVG) scores. For example, the MAX-aggregate scores of the 5 documents in Figure 2(a) are  $\{0.6, 0.95, 0.5, 1, 0.75\}$ . Although previously studied time-travel keyword queries share some similarity to durable top- $k$  search, they cannot directly be used for this new query.

We use a real example to show the special nature of durable top- $k$  search. We use a dataset from [22], which contains 11,328 URLs from Google Directory<sup>1</sup> and find their archive versions in 2004 at the Internet Archive. Consider a query with keywords *healthy* and *policy* and time interval the third and fourth quarters in 2004. We set  $k = 20$  and  $r = 0.5$  ( $r$  is tuned to ensure that the durable query produces the same number of results as other models). The number of different documents between the result of the durable top- $k$  query and the relevance models MIN, MAX, and AVG is 5, 8, and 8, respectively. The results of aggregate models are derived from the utmost/average scores, which are not directly related to consistency. Table 1 shows the different URLs computed by the MIN model and the durable query. DUR-MIN (MIN-DUR) contains the URLs which exist in the durable (MIN model) result but not in the MIN model (durable) result. Note that the URL [www.asgoodasnews.com](http://www.asgoodasnews.com) is a news magazine website and it discussed *healthy* and *policy* at some time in 2004 but not consistently. Therefore, it is not in the result of the durable query. On the other hand, the durable top- $k$  results are consistently relevant to the query within the given time interval, and exclude noisy outliers.

The main challenge in processing durable top- $k$  queries is that they are based on an ad-hoc set of multiple keywords. This means that the rankings of the document versions that overlap with the query interval are not pre-defined and can only be determined from the inverted lists of the query keywords. For example, the content of Figure 2(a) is not pre-computed, but dynamically derived

<sup>1</sup><http://directory.google.com/>

**Table 1: Example of durable top- $k$  search results**

DUR-MIN	MIN-DUR
<a href="http://iwhc.org">iwhc.org</a>	<a href="http://www.goodusedstuff.com">www.goodusedstuff.com</a>
<a href="http://www.4girls.gov">www.4girls.gov</a>	<a href="http://www.asgoodasnews.com">www.asgoodasnews.com</a>
<a href="http://research.aarp.org/health">research.aarp.org/health</a>	<a href="http://www.hsrnet.com">www.hsrnet.com</a>
<a href="http://www.accessexcellence.org">www.accessexcellence.org</a>	<a href="http://www.fasthealth.com/journals">www.fasthealth.com/journals</a>
<a href="http://www.luteininfo.com">www.luteininfo.com</a>	<a href="http://hometown.aol.com/lhcinc">hometown.aol.com/lhcinc</a>
<a href="http://www.homeinonhealth.com">www.homeinonhealth.com</a>	<a href="http://www.sierrahealth.com">www.sierrahealth.com</a>

by intersecting the inverted lists of the query keywords (these contain the document versions that include the keywords), skipping document versions that are outside the query time interval. For the keyword queries we consider in this work, documents are typically ranked using a relevance model such as Pivoted Normalization [23], Okapi BM25 [20], variants of tf-idf [24], or language modeling approaches [19]. The obtained relevance scores can be represented as sums of keyword-specific contributions. Top- $k$  aggregation techniques [11] are immediately applicable, if the relevance scores of document versions to each keyword are pre-computed and the versions are ordered in the corresponding inverted lists.

In this paper, we propose an efficient, specialized technique for durable top- $k$  queries. Our method is based on a storage organization, which sorts the contents of the inverted lists for each keyword in descending score order. While accessing the lists of the query keywords in parallel, our method maintains for each  $i$ ,  $1 \leq i \leq k$ , a *band* in the query timeline  $[t_b, t_e]$  capturing the scores of the current top- $i$  results for each timestamp in the query interval. At the same time, we maintain a *candidates band* capturing, for each timestamp in  $[t_b, t_e]$ , the best possible score of any object, which is not currently in the top- $k$  at that timestamp. If, during retrieval, the  $k$ -th band is above the candidates band, then the top- $k$  results at each timestamp are confirmed, which allows us to post-process them and identify the response set of the durable top- $k$  query. We pair our method with several optimization techniques that minimize the expensive maintenance of the candidates band and accelerate time-travel search with the help of spatial indexing. In addition, we propose a transformed R-tree index for indexing the inverted lists on the disk. With the help of this transformation, we are able to decompose a durable query into a set of simple top- $k$  queries and one durable query with smaller search space. This way, not only the I/O but also the computational cost is reduced. More interestingly, our durable top- $k$  search approach derives the top- $k$  results at every timestamp, before computing the durable result. Hence, durable results of different consistency can be found by progressively increasing the parameter  $r$ .

The rest of the paper is organized as follows. Section 2 describes work related to the problem under study, which is formally defined in Section 3. Section 4 presents a baseline approach to solve durable top- $k$  queries. Our optimized technique is described in Section 5. Section 6 describes an indexing technique that improves performance. Section 7 empirically evaluates our proposed solution using Wikipedia data. Finally, Section 8 concludes the paper and discusses future research directions.

## 2. BACKGROUND AND RELATED WORK

In this section, we review previous work which is closely related to our problem, such as top- $k$  search, indexing versioned documents, and time-travel queries in document databases.

## 2.1 Top- $k$ Queries

Fagin et al. [11] proposed and analyzed methods for top- $k$  merging of ranked lists, based on sorted and random accesses. In an Information Retrieval (IR) system, the relevance scores of a keyword to all documents (or document versions) are precomputed and stored in an inverted list. If the lists are ordered by score, we can apply the methods of [11] to find the most relevant documents (or versions) to a given set of keywords. As random accesses at inverted lists are significantly more expensive compared to sorted ones, we describe the “no-random accesses” (NRA) algorithm (Algorithm 1), which computes top- $k$  results using sorted accesses only. NRA iteratively retrieves objects  $o$  from the ranked inputs and maintains the upper  $\gamma_o^{ub}$  and lower bounds  $\gamma_o^{lb}$  of their aggregate scores. Bounds  $\gamma_o^{ub}$  and  $\gamma_o^{lb}$  are the atomic scores of  $o$  seen so far plus the highest and lowest possible score from the lists which have not been seen. Let  $W_k$  be the set of the  $k$  objects with the largest  $\gamma_o^{lb}$ . If the smallest value in  $W_k$  is at least the largest  $\gamma_o^{ub}$  of any object  $o$  not in  $W_k$ , then  $W_k$  is reported as the top- $k$  result and the algorithm terminates. LARA [18] is an efficient implementation of NRA, which manages the candidate results in a lattice and minimizes redundant bound computations and checks.

**Algorithm 1** NRA Algorithm

---

**NRA(sorted lists  $L$ )**

---

```

1: perform sequential accesses to each  $L_i$ ;
2: for each new object  $o$  update  $\gamma_o^{lb}$ ;           ▷ lower score bound
3: if less than  $k$  object have been seen so far then goto Line 1;
4: for each object  $o$  seen so far compute  $\gamma_o^{ub}$ ;       ▷ upper bound
5:  $W_k :=$  the  $k$  objects with the highest  $\gamma_o^{lb}$ ;
6:  $l := \min\{\gamma_o^{lb} : o \in W_k\}$ 
7:  $u := \max\{\gamma_o^{ub} : o \notin W_k\}$ 
8: if  $l \geq u$  then return  $W_k$ ; otherwise goto Line 1;
```

---

[16] is the most relevant piece of work to durable top- $k$  search. Given a database of time-series (spanning the same history) and a time interval  $[t_b, t_e]$  (which is contained in the history), the problem is to find the time-series that are consistently in the top- $k$  set for each timestamp of the query interval. For example, assuming a database of stock transactions, one might want to find stocks that are consistently in the top-20 by turnover, during the first three months of 2009. There are certain differences between this problem to the durable top- $k$  queries that we study here. First, the contents of the time-series in [16] are pre-defined (i.e., not ad-hoc), therefore the values on which the top- $k$  function is applied are pre-computed and can be indexed. On the other hand, in our problem, ranking is defined based on an ad-hoc set of keywords. Although relevance with respect to a single keyword is pre-defined, pre-processing and indexing for an ad-hoc keyword combination is not possible. Second, the definition of [16] lacks the parameter  $r$ , which relaxes the definition of durability and avoids otherwise empty query results (i.e., for  $r = 1$ ). Finally, the solution suggested in [16] does not scale well. For each object  $o$ , a sequence of ranks for  $o$  in the whole history (e.g., 1st at the 1st timestamp, 3rd at the 2nd timestamp, etc.) is pre-computed. To find consistent top- $k$  objects during a time interval, we search within each object list for its ranks in the interval and we output only those documents whose ranks in all orderings in the time interval are at most  $k$ . The cost of this method is proportional to the number of objects (i.e., a time-interval search first should be applied at each object list, then the retrieved ranks have to be accessed and compared with  $k$ ), so the method does not scale well with the number of objects.

Pruning strategies in IR, based on specially designed inverted files, were proposed in [1, 2, 26]. The contents of the inverted lists can be ordered by document ids or scores. The first approach is the

classic implementation, which enables multi-way merging when processing a query with multiple keywords. The sizes of inverted lists can be reduced by storing the difference (i.e., gap) between consecutive ids. If the inverted lists are stored in this way, a simple solution could be used to compute durable top- $k$  queries. First, we traverse the complete inverted lists and put all entries that intersect the query interval into a temporary array. Next, each record in the temporary array is split into a start and end event and all events are sorted by time. The durable top- $k$  result is computed by running a simple scanline algorithm from computational geometry. The scanline is run on a heap of size  $k$  maintaining the current top- $k$ , with an additional data structure for elements that have been in the top- $k$  but are currently not. Such an approach is not efficient in practice, due to the large overhead of generating, sorting, and scanning a potentially huge number of document versions.

The second approach used in [1] keeps document entries in the inverted lists ordered by scores and uses Algorithm 1 to terminate list intersection early. However, the inverted lists cannot be easily compressed, since the scores may be in floating point format. In view of this, a hybrid approach is proposed in [2, 26], which uses a two level structure. The documents in a list are decomposed into different segments based on their scores and the documents in one segment are sorted by ids to facilitate compression. NRA used also on this data representation, but this time all segments having the same score bounds are accessed in a batch and after each batch access the termination condition is verified.

## 2.2 Indexing versioned document collections

Indexing versioned document collections has been studied in [7, 25, 14, 13]. Broder et al. [7] propose a technique that exploits large content overlaps between documents to achieve a reduction in index size. Each version is partitioned into a set of fragments, e.g., an email is partitioned into two fragments, subject and body. The fragments between versions are organized in a tree structure and each child inherits the shared fragments from its parent. This solution makes strong assumptions about the structure of document overlaps. [25] uses content-dependent partitioning technique [21] to partition a page into smaller fragments such that more fragments are common between versions. More recent approaches by Hersovici et al. [14] and He et al. [13] exploit arbitrary content overlaps between documents to reduce index size. [14] attempt to find subsets of terms that are contained in consecutive versions of a document. Each subset is stored into a virtual document and the total storage cost is optimized by minimizing the overall number and size of the virtual documents. [13] propose a two-level index compression that improves the query processing time. This approach groups similar union-documents into clusters, where a union-document contains all terms in the corresponding versions, and the terms are compressed locally for each cluster. This structure greatly reduces storage and still preserves the hierarchical relationship between documents and versions.

All these indexing methods primarily aim at reducing the space required for storing the versioned document collections, taking advantage of the similarity between versions. However, the durable top- $k$  search problem that we study in this paper is CPU-intensive and does not benefit directly from such compression techniques, since all versions may have to be accessed and reconstructed from the compressed storage scheme.

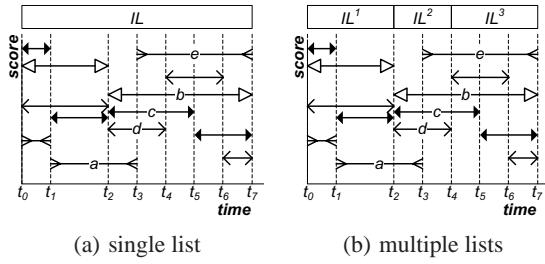
## 2.3 Time Travel Queries in IR

There is a significant body of work on analyzing large text collections over time. Bansal and Koudas [4] describe a full-fledged system for searching the blogosphere. Among others, the system sup-

ports the detection of stable keyword clusters as described in [3]. Earlier work by Kleinberg [15] and Dubinko et al. [10] also focuses on the analysis of text and tag streams, respectively, to detect bursty keywords or tags. However, all of the three aforementioned approaches operate on the document collection as a whole and not on ad-hoc keyword query results.

Other work has investigated the use of temporal information as a means to obtain a better ranking of query results. Li and Croft [17], for instance, propose a language modeling approach that factors in the publication of the document. Del Corso et al. [8] focus on news and propose ranking methods that take into account when a news article was published and linked to by other news articles.

Berberich et. al. [5] proposed a temporal text indexing technique for web documents which supports time-travel queries. In a typical inverted file, each inverted list contains a *posting*  $(d, s)$ , where  $d$  is the document id and  $s$  is the relevance of the term in document  $d$ . To support indexing of versioned documents, in [5], the inverted file is extended, such that each posting includes a time interval  $\lambda$ . The temporal information characterizes the validity time interval of the indexed version of  $d$ . The objective of a time-travel query is to identify the top- $k$  documents with the highest aggregate score during the query interval, as we explained in the Introduction.



**Figure 3: Comparison between one and multiple inverted lists**

Consider a set of postings for a keyword  $w$  and a top- $k$  time-travel query  $q$  on  $w$  with time interval  $[t_2, t_3]$  as shown in Figure 3(a). The query can be processed by accessing the postings in decreasing score order, ignoring those that do not overlap with the query interval. While doing so, we can use upper bounds for the subsets of the query interval where postings have not been seen (i.e., run a version of NRA) and at some point confirm the  $k$  documents with the best aggregate scores. Although only four postings ( $a, b, c$ , and  $d$ ) are valid in time  $[t_2, t_3]$ , the whole inverted list has to be read, in the worst case. To tackle this problem, Berberich et. al. [5] propose a partitioning approach, which splits the inverted list with the entire posting set into smaller lists. For instance, we can partition the inverted list of Figure 3(a) into three sub-lists as shown in Figure 3(b). Each posting is stored in all lists which it temporally intersects (e.g., posting  $a$  is stored into  $IL^1$  and  $IL^2$ ). Now, query  $q$  temporally intersects only with list  $IL^2$ , therefore only five postings have to be read (instead of 13 if  $IL$  of Figure 3(a) is used).

One strategy is to materialize sub-lists for all elementary time intervals. For instance, we could create 7 sub-lists for the data in Figure 3(a). This achieves excellent performance for queries with short intervals, but a lot of space is wasted due to replicated storage of postings that intersect multiple list intervals. In addition, queries with long time intervals access multiple lists with overlapping contents. In view of this, [5] study the optimization problem of splitting the lists to a suitable set of sub-lists with or without a constraint for the space occupied by them.

### 3. PROBLEM DEFINITION

Problem 1 is a formal definition of the durable top- $k$  query. Although this definition is tailored for temporal keyword search in archives of documents with versions, we can adapt it (and the solutions proposed in this paper) to apply on any type of data, with different versions along arbitrary dimensions (e.g., document versions based on location, or blog items grouped by user).

**PROBLEM 1.** Let  $\mathcal{D}$  be a set of  $n$  documents. Each  $d \in \mathcal{D}$  has a number of versions, and each version  $v_d$  of  $d$  is characterized by a validity time interval  $[v_d.t_b, v_d.t_e]$ . The time intervals of two different versions of the same document may not overlap. Let  $q$  be a query, consisting by a set of keywords  $q.W$  and a time interval  $[q.t_b, q.t_e]$ . For a timestamp  $t \in [q.t_b, q.t_e]$ , the relevance of a document  $d \in \mathcal{D}$  to  $q$  is defined by applying an IR relevance model on the version  $v_d$  of  $d$  for which  $t \in [v_d.t_b, v_d.t_e]$ , using  $q.W$ . The relevance is zero if no such version exists. Given an integer  $k$ ,  $0 < k < n$  and a real  $r$ ,  $0 < r \leq 1$ , the durable top- $k$  search problem finds all  $d \in \mathcal{D}$ , such that  $d$  appears in the set of top- $k$  most relevant documents to  $q$  within  $[q.t_b, q.t_e]$  for time at least  $r \times (q.t_e - q.t_b)$ .

### 4. PRELIMINARY SOLUTIONS

In this section, we describe some direct adaptations of NRA for solving the durable top- $k$  search problem. For the ease of discussion, we assume that all postings for each keyword are sorted by their scores and materialized into a single inverted list. The use of multiple inverted lists per keyword (as proposed in [5]) is orthogonal to the presented solutions and will be discussed later.

#### 4.1 Brute-force method

Consider a query  $q$  with a set of keywords  $W$  and a time interval  $q.\lambda = [q.t_b, q.t_e]$ . Let  $\Lambda$  denote a set of sub-intervals such that (i) no two intervals in  $\Lambda$  overlap, (ii) their union equals  $[q.t_b, q.t_e]$  and (iii) each document version either fully covers or does not overlap with any sub-interval in  $\Lambda$ . Then, finding the top- $k$  results in each sub-interval suffices to compute the durable top- $k$  result. Condition (iii) ensures the uniqueness of each document in a sub-interval and conditions (i) and (ii) guarantee completeness. After collecting the top- $k$  results from all sub-intervals, we intersect them, while measuring for each document  $d$  the total temporal length of the sub-intervals where  $d$  is in the top- $k$  result. If this length is at least  $r \times (t_e - t_b)$ ,  $d$  is a durable top- $k$  result.

Algorithm 2 is a greedy method that finds  $\Lambda$  incrementally, by accessing the postings that intersect with  $q.\lambda$ . Initially, we set  $\Lambda = q.\lambda = [q.t_b, q.t_e]$ . For each posting  $p$  with interval  $[p.t_b, p.t_e]$ , we find the subset  $\Lambda'$  of  $\Lambda$  such that all intervals in  $\Lambda'$  intersect  $p$  (line 1). If the begin/end timestamp  $p.t$  of  $p$  is inside of an interval  $[t_b, t_e]$  in  $\Lambda'$ , this interval is split and replaced by the two intervals  $[t_b, p.t]$  and  $[p.t, t_e]$  (line 3). We can easily show that this algorithm computes a unique correct set of (max-length) sub-intervals that satisfy conditions (i), (ii), and (iii). We can then compute the top- $k$  results within each interval and the durable top- $k$  result.

---

#### Algorithm 2 Interval Set Maintenance

---

```

maintainIS(interval set  $\Lambda$ , new interval  $[p.t_b, p.t_e]$ )
1:  $\Lambda'$  is a subset of  $\Lambda$  that each interval intersects with  $[p.t_b, p.t_e]$ ;
2: if  $p.t_b$  or  $p.t_e$  is inside a  $[t_b, t_e] \in \Lambda'$  then
3:   replace  $[t_b, t_e]$  by two sub-intervals;

```

---

For example, assume that our query contains 1 keyword and the inverted list contains four postings  $a, b, c$ , and  $d$ , as shown in Figure 4. Postings  $a$  splits the entire interval  $\lambda_0 = [q.t_b, q.t_e]$  into two  $\lambda_1$

and  $\lambda_2$  ( $\lambda_2$  is the union of  $\lambda_3$  and  $\lambda_4$ , shown in the figure). Next, posting  $b$  intersects interval  $\lambda_2$  (i.e.,  $\Lambda' = \{\lambda_2\}$ ).  $\lambda_3$  and  $\lambda_4$  are created since  $b.t_b$  is inside of  $\lambda_2$ . Next, posting  $c$  intersects intervals  $\lambda_1$ ,  $\lambda_3$ , and  $\lambda_4$ . The starting point of  $c$  is not inside  $\lambda_1$ , so  $\lambda_1$  is not split to sub-intervals. On the other hand, the endpoint of  $c$  is contained in  $\lambda_4$ , so  $\lambda_4$  is replaced by new sub-intervals  $\lambda_5$  and  $\lambda_6$  (see Figure 4(b)). In turn, posting  $d$  splits  $\lambda_3$  and  $\lambda_5$ , ending with 6 sub-intervals in total (not shown in the figure).

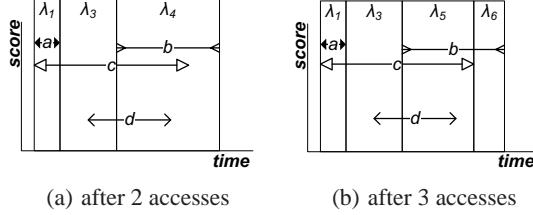


Figure 4: Example of intervals maintenance

## 4.2 Dynamic Adaptive Algorithm

The brute-force solution is inefficient since (i) it reads all postings that intersect the query interval to create the sub-intervals set and (ii) many sub-intervals are created, which require a large number of top- $k$  queries. For instance, we have 6 sub-intervals after we process 4 postings in Figure 4, meaning that 6 top- $k$  searches should be executed before we can collect the durable top- $k$  result. Note that some sub-intervals need not be computed at all if  $k$  is small. For instance, if  $k = 1$ , we can terminate our search after posting  $c$  is read since we can find the top-1 result for each time interval already. As shown by Algorithm 2, the sub-intervals can be maintained incrementally. Thus, we can maintain the sub-intervals and execute top- $k$  aggregation simultaneously.

For each sub-interval, NRA is invoked to compute the top- $k$  result. According to Algorithm 1, we have to keep the lower bound  $\gamma_o^{lb}$  and the upper bound  $\gamma_o^{ub}$  for every object  $o$  seen so far. Let  $\Gamma_\lambda^{lb}$  and  $\Gamma_\lambda^{ub}$  be the set of lower and upper bounds in interval  $\lambda$ , respectively. The usage of these bound sets will be discussed shortly.

---

### Algorithm 3 Dynamic Adaptive Algorithm

---

```

DAA(sorted posting lists  $L$ )
1:  $p :=$ access the next posting from  $L$ ;
2: maintainIS( $\Lambda$ ,  $[p.t_b, p.t_e)$ );
3: for all new  $\lambda \in \Lambda$  do
4:   if  $\lambda$  is finalized, goto line 3;
5:   create or duplicate  $\Gamma_\lambda^{lb}$  and  $\Gamma_\lambda^{ub}$  for interval  $\lambda$ 
6:   if intersects( $\lambda$ ,  $[p.t_b, p.t_e)$ ) then
7:     feed  $p$  to NRA for  $\lambda$  using  $\Gamma_\lambda^{lb}$  and  $\Gamma_\lambda^{ub}$ ;
8:     if NRA returns top- $k$  result, mark  $\lambda$  as finalized;
9: if all  $\lambda \in \Lambda$  are finalized then
10:   compute durable top- $k$  result;
11: else
12:   goto Line 1;

```

---

According to Algorithm 2, if more postings are read from the lists, more sub-intervals are created. Therefore, we propose a technique, called Dynamic Adaptive Algorithm (DAA), to terminate our search as early as possible (see Algorithm 3). This is possible, if the postings in the lists are sorted in descending score order. First, we access the postings sequentially and the sub-intervals are maintained by Algorithm 2. If an existing interval is split into two new ones, because it contains one endpoint of the currently processed posting  $p$ , then for each of the two new intervals  $\lambda$ ,  $\Gamma_\lambda^{lb}$  and  $\Gamma_\lambda^{ub}$  are replicated from the old split interval.

After the sets  $\Gamma_\lambda^{lb}$  and  $\Gamma_\lambda^{ub}$  are created for the new interval  $\lambda$ , we use the current posting  $p.d$  as the next input to NRA to update the bounds and the current top- $k$  results.<sup>2</sup> If NRA confirms the top- $k$  result in  $\lambda$ , we mark interval  $\lambda$  as “finalized”. That is, no further splits are performed to  $\lambda$ , if new postings are found to intersect it later. If all intervals are marked as *finalized*, we merge their top- $k$  results to compute the durable top- $k$  set. Otherwise, we get the next posting from the inverted lists.

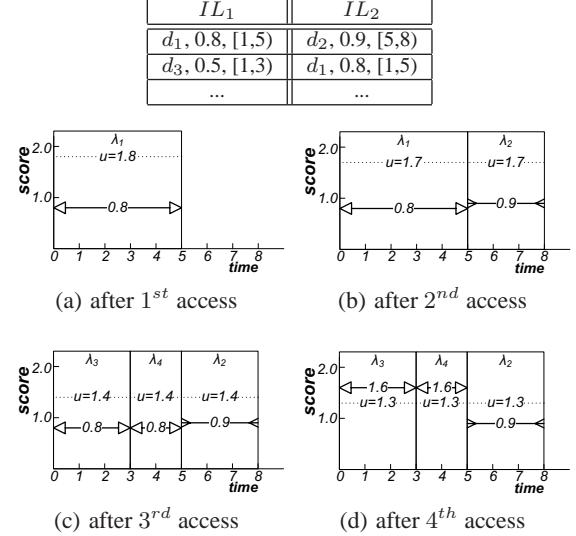


Figure 5: An example of Dynamic Adaptive Algorithm

Figure 5 demonstrates the Dynamic Adaptive Algorithm. The postings of two keywords stored into two inverted lists  $IL_1$  and  $IL_2$ , as shown at the top of the figure. Assume that  $k = 1$ . After the first access,  $\Gamma_{\lambda_1}^{lb}$  and  $\Gamma_{\lambda_1}^{ub}$  are created.  $d_1$  is currently in the top- $k$  set  $W_k$  of  $\lambda_1$  and  $u$  is 1.8 in  $\lambda_1$  (note that  $u$  is the highest upper bound score of any  $o \notin W_k$ ). Next, we get  $d_2$  from  $IL_2$  and we create  $\Gamma_{\lambda_2}^{lb}$  and  $\Gamma_{\lambda_2}^{ub}$ .  $u$  in both intervals are 1.7 now. In order to improve readability, we remove all data which are not in  $W_k$  in the subsequent figures. After the third access ( $d_3$  from  $IL_1$ ),  $\lambda_1$  is split into two intervals  $\lambda_3$  and  $\lambda_4$  and their  $\Gamma^{ub}$  and  $\Gamma^{lb}$  are duplicated from  $\lambda_1$ .  $u$  in  $\lambda_3$  is 1.4 since  $d_3$  has been seen in this interval with the upper bound  $\gamma_{d_3}^{ub}$  ( $0.5+0.9=1.4$ );  $u$  in other intervals is also 1.4, which is the highest aggregate score from all lists. Finally, after the fourth access, we update the  $d_1$ 's  $\gamma^{lb}$  to 1.6 in  $\lambda_3$  and  $\lambda_4$ . In addition,  $u$  becomes 1.3 in all intervals. According to the NRA termination condition, intervals  $\lambda_3$  and  $\lambda_4$  return  $d_1$  as their top-1 result.

## 5. THE BAND APPROACH

During the execution of DAA, many  $\Gamma_\lambda^{lb}$  and  $\Gamma_\lambda^{ub}$  sets are created. These affect negatively not only the execution time but also the memory usage. In this section, we introduce a new method that solves the durable top- $k$  problem using the shared execution paradigm, based on the observation that two neighbor intervals usually have similar top- $k$  results. In a nutshell, our method performs similar splits as DAA, however, we do not maintain  $\Gamma_\lambda^{lb}$  and  $\Gamma_\lambda^{ub}$  at each sub-interval. Instead, we maintain (in a compressed representation) for each  $i$ ,  $1 \leq i \leq k$ , the band (i.e., boundary) for the  $i$ -th worst-case score at each time unit in the query interval. In addition, we

<sup>2</sup>In case of a split, the new posting is fed to only one of the two intervals: the one that intersects the posting.

maintain in a *candidates band* the best-case score of all objects currently not in the top- $k$  set for each time unit. If the candidates band drops below the  $k$ -th band at all time units, we can guarantee that the top- $k$  results are found at all timestamps and we can terminate.

## 5.1 Top Bands Computation

First, we define the concept of top- $k$  band. Consider the finest granularity unit of the time dimension (e.g., days) and assume that posting  $p$  spans  $u$  units of this granularity (i.e., the time interval  $[p.t_s, p.t_e]$  includes  $u$  basic time units). Then,  $p$  can be modeled as a sequence of  $u$  postings  $p = \tau_p^1, \tau_p^2, \dots, \tau_p^u$ , each spanning a single time unit. Note that the other attributes (i.e., document id and score) are common to all unit-postings of  $p$ . Assuming a representation, where each posting is replaced by its unit-postings, Definition 1 formally defines the top- $k$  band.

**DEFINITION 1.** *The top- $k$  band is a sequence of  $\tau_p^t$  unit-postings, one for each time unit  $t$  in  $[q.t_s, q.t_e]$ , where  $\tau_p^t$  is the unit-posting among all those valid at time  $t$  with the  $k$ -th highest score.*

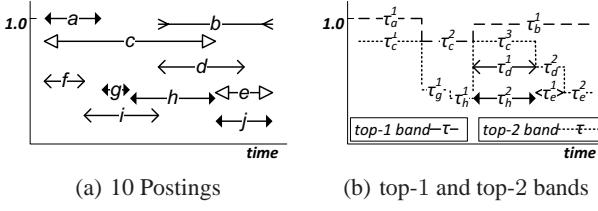


Figure 6: An example of top- $k$  bands

Figure 6 shows an example of top- $k$  bands. Consider a keyword with 10 postings ( $a$  to  $j$ ), as shown in Figure 6(a). The top-1 and top-2 bands are shown in Figure 6(b). The postings that are not related to the top-1 and top-2 bands are removed. For simplicity, we overload the notation  $\tau_p^i$  to denote segments of consecutive unit-postings from the same posting  $p$  and use  $i$  to distinguish between segments from the same posting  $p$ . The top-1 band consists of 3 segments  $\{\tau_a^1, \tau_c^1, \tau_b^1\}$ . Note that  $\tau_a^1$  and  $\tau_b^1$  are the same as postings  $a$  or  $b$  respectively since there is no other segment better than  $a$  or  $b$  in their valid intervals. Posting  $c$  is decomposed into three segments and only  $\tau_c^2$  appears on the top-1 band. By removing all segments in the top-1 band (i.e.,  $\tau_a^1, \tau_c^1$ , and  $\tau_b^1$ ), the top-2 band becomes the top-1 band of the remaining segments. This important property helps us to maintain the bands iteratively.

Algorithm 4 shows a recursive method that maintains the set of top- $k$  bands for single keyword queries (multiple keywords queries will be discussed shortly).  $T_i^{top}$  denotes the top- $i$  band and it is modeled by a set of segments. Let  $T^{ins}$  be a set of new segments that should be used to update the top bands. Each  $\tau \in T^{ins}$  is decomposed into a set of small segments  $\tau = \{\tau^1 \cup \dots \cup \tau^n\}$ , according to  $\tau$ 's intersections with  $T_i^{top}$ . For instance, in Figure 7, posting  $d$  is decomposed into 3 segments  $\{\tau_d^1, \tau_d^2, \tau_d^3\}$ .

After we collect the set of segments  $\{\tau^1, \dots, \tau^n\}$ , we compare them with the top- $i$  band  $T_i^{top}$  one at a time. If  $\tau^i$  does not intersect with any segment in  $T_i^{top}$  (line 5),  $\tau^i$  is inserted into  $T_i^{top}$  since it fills a current gap in the top- $i$  band. Suppose that there is a  $\tau^{top}$  intersecting with  $\tau^i$  at  $t$ . If the score of  $\tau^{top}$  is not worse than the score of  $\tau^i$ ,  $\tau^i$  is moved to  $T^{next}$  to be processed in the next band (line 7). For instance,  $\tau_d^1$  in Figure 7(b) will be processed in the top-2 band. If  $\tau^i$  has a higher score than  $\tau^{top}$ , we decompose  $\tau^{top}$  at time  $t$ , which is the endpoint of  $\tau^i$  inside  $\tau^{top}$ . Consider again the example of Figure 7(b). Segment  $\tau_d^1$  intersects with  $\tau_d^3$  and its score is lower than the score of  $\tau_d^3$ . Therefore, it is decomposed into  $\{\tau_d^2, \tau_d^3\}$ .  $\tau_d^3$  and  $\tau_d^2$  have the same validity interval ( $\tau_d^3.\lambda = \tau_d^2.\lambda$ ).

---

### Algorithm 4 Top- $k$ Bands Insertion

---

```

insertBand(top- $i$  band  $T_i^{top}$ , set of segments  $T^{ins}$ , int  $k$ )
1: set  $T^{next} := \emptyset$ 
2: for  $\tau \in T^{ins}$  do
3:   for each  $\tau^{top} \in T_i^{top}$ , decompose  $\tau$  if  $\tau \cap \tau^{top} \neq \emptyset$ , such that  $\tau$  is
      decomposed into  $\{\tau^1, \dots, \tau^n\}$ 
4:   for each  $\tau^i$  do
5:     if not intersect( $\tau^i, \tau^{top}$ ) then
6:       insert into  $T_i^{top}$ 
7:     else if  $\tau^{top}.s \geq \tau^i.s$  then
8:       insert  $\tau^i$  into  $T^{next}$ ;
9:     else ▷ intersect at t
10:      use  $t$  to decompose  $\tau^{top}$  into  $\tau^{top1}$  and  $\tau^{top2}$ 
11:      # assuming that  $\tau^{top1}.\lambda = \tau^i.\lambda$ 
12:      replace  $\tau^{top}$  by  $\tau^{top2}$  and  $\tau^i$ 
13:      insert  $\tau^{top1}$  into  $T^{next}$ ;
14:  $T^{ins} := T^{next}$ ;
15: if  $i < k$  then
16:   return insertBand( $T_{i+1}^{top}, T^{ins}, k$ );
17: else
18:   return  $T^{ins}$ ;

```

---

Based on their scores,  $\tau_d^3$  and  $\tau_c^3$  are inserted into the top-1 band and  $\tau_d^2$  is stored in  $T^{next}$  to be processed at the next call of the algorithm (line 16), which computes the next band.

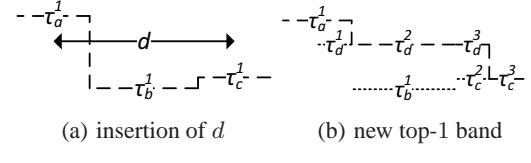


Figure 7: Insertion in top- $k$  band

This process guarantees that the top- $k$  results at all timestamps are equivalent to the results of the top bands. Once we collect all top bands, we process to compute the durable top- $k$  result.

So far, we have discussed how to maintain the top bands for single keyword queries. If we have multiple keywords, we read the postings from each keyword (in parallel) and insert each of them into the top bands using Algorithm 4. If the document version in the current posting has been seen at some other keyword list before and it is in the top- $i$  band, we remove it from the band and reinsert it considering its updated relevance score (which is increased compared to its previous partial score). Note that the removal process does not require to access any data outside the top- $k$  bands. More precisely, this does not require any change at the top- $(i+1)$  to top- $k$  bands since the relevance score is only increasing.

## 5.2 Candidate Band Computation

The simplest way to compute top- $k$  bands is to read all postings from each keyword once and exhaustively, while updating the bands at each reading. However, we would like to terminate the accesses to the inverted lists as early as possible. Therefore, we investigate an appropriate access plan and termination condition for our band approach. If we access the postings in decreasing order of their scores (i.e., like DAA), a termination condition can be derived with the help of a typical top- $k$  aggregation approach. Before we discuss our solution, we define the concepts of *candidate container*  $C$  and *candidate band* in Definitions 2 and 3.

**DEFINITION 2.** *The candidate container  $C$  stores the segments of document versions which have been seen so far but they are not included in top bands.*

DEFINITION 3. The candidate band contains the segments that are the top-1 band of the candidate container  $C$ .

Note that the upper bound  $u$  in NRA is computed by the highest possible scores  $\gamma_o^{ub}$  from the objects that are not in  $W_k$  (where  $W_k$  contains the  $k$  objects with the highest  $\gamma_o^{lb}$ ). Similarly, we store all segments that are not in our top bands to our candidate container  $C$ . For each segment  $\tau^i$  in  $C$ , its score is set to be the upper bound  $\gamma^{ub}$  of  $\tau^i$ . We compute a top-1 band from  $C$  and this is our *candidate band*. After every access, if we have the top- $k$  band and the candidate band, it is possible to terminate retrieval if the conditions of Lemma 1 are satisfied for every timestamp in the query interval.

LEMMA 1. The top- $k$  result at timestamp  $t$  has been stored in the top bands if and only if the score of the  $k$ -th band at time  $t$  is not worse than (1) the score of the candidate band  $C$  at time  $t$  and (2) the sum  $\Psi$  of the last seen scores at all lists. Condition (2) is checked if there is no element in  $C$  that intersects  $t$ .

Algorithm 5 shows the pseudocode of our band approach. We use a set operation ' $\geq_{\forall \lambda}$ ' to denote comparisons over a time interval  $\lambda$ . For instance,  $T^{top} \geq_{\forall \lambda} T^{cand}$  means that at any timestamp  $t$  in  $\lambda$  the segments in  $T^{top}$  that are valid at  $t$  are not worse than the segments in  $T^{cand}$  that are valid at  $t$ . Similarly,  $T^{top} \geq_{\forall \lambda} \Psi$  denotes that the segments in  $T^{top}$  are not worse than the line defined by  $\Psi$ . For the current posting, if a segment has been inserted into the top bands or the candidate container, we remove it and reinsert it with an updated score (lines 2-4). Then, subroutine *insertBand* is called (Algorithm 4), which updates the top bands incrementally by inserting  $T^{ins}$  and returns the already-seen set of segments  $T^{ret}$  that are outside the top bands (line 5). Next, we insert all segments in  $T^{ret}$  into the candidate container  $C$  and compute the candidate band (lines 6-9). Finally, if all timestamps meet the conditions of Lemma 1, we terminate our search and proceed to find the durable top- $k$  result using the top bands.

#### Algorithm 5 Band Based Algorithm

---

BBA(sorted lists  $L$ )

- 1:  $p :=$ access the next posting from  $L$ ;
- 2: **for all**  $\tau^i \in p$  **do**
- 3:   remove  $\tau^i$  from top band / candidate container;
- 4:   set  $\tau^i.s$  to its  $\gamma^{lb}$  and insert into  $T^{ins}$ ;                      ▷ use lower bound
- 5:  $T^{ret} := \text{insertBand}(T_1^{top}, T^{ins}, k)$ ;                      ▷ top bands
- 6: **for all**  $\tau^j \in T^{ret}$  **do**
- 7:   insert  $\tau^j$  into  $C$ ;
- 8: set  $\tau^j.s$  to its  $\gamma^{ub}$ ,  $\forall \tau^j \in C$ ;                              ▷ use upper bound
- 9:  $T_1^{cand} := \text{compute top-1 band from } C$ ;                      ▷ candidate band
- 10: **if**  $T_k^{top} \geq_{\forall q, \lambda} T_1^{cand}$  and  $T_k^{top} \geq_{\forall q, \lambda} \Psi$  **then**
- 11:   check durable top- $k$  result;
- 12: **else**
- 13:   goto Line 1;

---

We use the data from Figure 5 to demonstrate our band approach, for  $k = 1$ . The first two postings (in round-robin order from the lists) are inserted into the top-1 band using line 5 and  $\Psi$  becomes  $0.8 + 0.9 = 1.7$ . When we insert the third posting, it fails to enter the top-1 band since its lower bound  $\gamma^{lb} = 0.5$  is not better than the  $\gamma^{lb} = 0.8$  of  $\tau_{d1}^1$ . Therefore, subroutine *insertBand* returns  $\{\tau_{d3}^1\}$ . After that, we insert it into the candidate container  $C = \{\tau_{d3}^1\}$ , and we set  $\tau_{d3}^1$ 's score to upper bound  $\gamma^{ub} = 0.5 + 0.9 = 1.4$ . The candidate band is then computed, as shown in Figure 8(c). When the fourth posting is read, we remove  $\tau_{d1}^1$  from the top-1 band since the score of  $\tau_{d1}^1$  is updated by this access. We update its score to  $0.8 + 0.8 = 1.6$  and reinsert it into the top

bands. Next, we recompute our candidate band, as the score of  $\tau_{d3}^1$  becomes  $0.5 + 0.8 = 1.3$ . Note that top-1 result in interval  $[1, 5]$  has been confirmed since the conditions ( $T^{top} \geq_{[1, 5]} T^{cand}$  and  $T^{top} \geq_{[1, 5]} \Psi$ ) become true. The algorithm will terminate later, after the interval  $[5, 8]$  is confirmed.

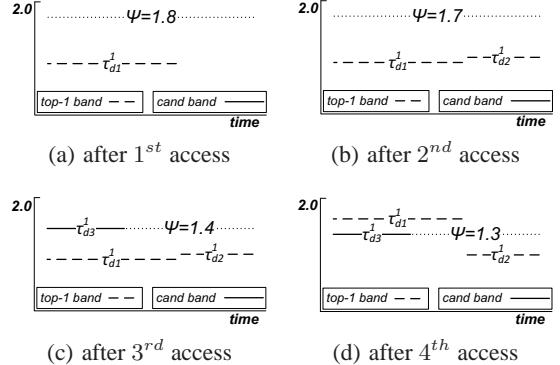


Figure 8: Examples of Band Based Algorithm

### 5.3 Optimizations

Note that the main difference between the *top bands* and the *candidate band* is that the lower bound  $\gamma^{lb}$  is used in the *top bands* but we use the upper bound  $\gamma^{ub}$  in the *candidate band*. Unlike the segments in *top bands*, the entire *candidate band* could be changed when a new posting is read from a list due to the changes of upper bounds. Consequently, the *candidate band* is recomputed at each loop of Algorithm 5. The recomputation of the *candidate band* might be very expensive when container  $C$  becomes very large. To reduce this cost, we use a powerset-based approach to support incremental updates at the *candidate band*. In addition, we propose a grid-based index to manage the data in the candidate container.

#### 5.3.1 Lattice Based Containers

Similar to LARA [18], we create a set of containers  $C_x$ , one for each combination of the  $m$  inputs  $\{L_1, \dots, L_m\}$  (recall that each input is the inverted list of a keyword). For each segment  $\tau^i$  in  $C$ ,  $\tau^i$  is stored into  $C_x$  if  $\tau^i$  has been seen exactly in the  $x$  inputs. The practical difference between  $C$  and the collection of  $C_x$ 's is that we maintain the lower bounds  $\gamma^{lb}$  of the segments in each  $C_x$ , as opposed to maintaining the upper bounds of all segments in  $C$ .

As for the top bands, the top-1 band  $T_1^{C_x}$  for each  $C_x$  can be computed incrementally. That is, when we insert a new segment  $\tau^i$  into  $C_x$ , if this segment has been seen by other lists, we remove it from its previous container  $C_y$  and  $T_1^{C_y}$ . (Note that after the removal of  $\tau^i$  from  $C_y$ ,  $C_y$  should be updated appropriately. This will be discussed in next subsection.) Finally, we insert  $\tau^i$  with an updated score  $\gamma^{lb}$  into  $C_x$  and call subroutine *insertBand* to maintain the top-1 band of  $C_x$ .

Now, let us see how we can derive the *candidate band* in  $C$  and use it in BBA. This can be easily done by merging the candidate bands of all  $C_x$ 's. We concurrently traverse these bands in time-order and for each  $C_x$  we add to the score of its top-band segment and the sum of the last scores from the lists  $L_d$  where  $d \notin x$ . Then we dynamically derive an upper bound for the segments of  $C_x$ . The dynamically derived upper bounds for each  $C_x$  are compared to compute the globally highest upper bound in all  $C_x$ 's at each timestamp. This is equivalent to the *candidate band* of  $C$ .

#### 5.3.2 Grid-Based Segments Management

If we do not manage the segments in  $C_x$  properly, we might have to access all segments in  $C_x$  each time a segment from  $C_x$

has to be moved to another candidate set (because it has been seen at a new input) and the candidate band for  $C_x$  has to be updated. To perform this operation efficiently, we use a grid index to divide the two dimensional time/score space for each  $C_x$  into cells. A segment is assigned to a cell if it intersects it. An example is shown in Figure 9(a). For instance, segment  $\tau_{d3}^1$  is inserted into 6 cells of the first row. If a segment  $\tau$  is deleted from  $C_x$ , it is removed from the corresponding cells. If  $\tau$  was part of  $C_x$ 's top band, we must seek for a replacement in the band. To do this, we first check the cells that intersect  $\tau$ , if they are empty the cells below, etc.

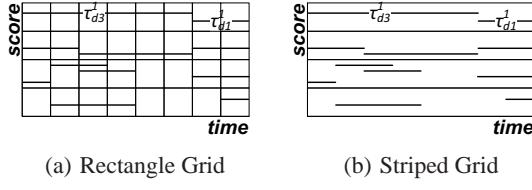


Figure 9: Examples of different Grid Indices

In our implementation, in order to avoid the replication of segments to multiple cells, we use a grid with only horizontal stripes such as that shown in Figure 9(b). As segments are horizontal, no segment is replicated. Updates are also more efficient in this case. In order to support fast search during top band updates at  $C_x$ , we order the segments within each cell in order to locate fast the ones that overlap the removed segment.

#### 5.4 Optimized Band Based Algorithm

An optimized version of BBA, which includes all optimizations of Section 5.3, is shown in Algorithm 6. The main changes from BBA to OBBA are lines 3-5 and 9-12. Once a segment  $\tau^i$  is removed from candidate container  $C_y$ , we access the corresponding grid cell of  $C_y$  to update its band (lines 4-5). Line 11 calls subroutine *insertBand* to compute the top-1 band of  $C_x$ , where  $\tau^i$  is inserted. Finally, the complete candidate band is computed by the set of candidate bands (line 12).

---

##### Algorithm 6 Optimized Band Based Algorithm

---

```

OBBA(sorted lists  $L$ )
1:  $p :=$ access the next posting from  $L_d$ ;
2: for all  $\tau^i \in p$  do
3:   remove  $\tau^i$  from top band / candidate band;
4:   if  $\tau^i$  is in candidate container  $C_y$  then
5:     remove  $\tau^i$  and replace interval of  $\tau^i$  using  $C_y$  grid;
6:     set  $\tau^i.s$  to its  $\gamma^{lb}$  and insert into  $T^{ins}$ ;            $\triangleright$  use lower bound
7:      $T^{ret} := \text{insertBand}(T_1^{top}, T^{ins}, k)$ ;            $\triangleright$  top bands
8:   for all  $\tau^j \in T^{ret}$  do
9:     set  $\tau^j.s$  to its  $\gamma^{lb}$ ;            $\triangleright$  use lower bound
10:    insert  $\tau^j$  into  $C_x$ ;
11:     $\text{insertBand}(T_1^{C_x}, T^{ret}, 1)$ ;            $\triangleright$  top-1 band of  $C_x$ ;
12:    compute  $T_1^{cand}$  using  $\{T_1^{C_1}, \dots, T_1^{C_{2m}}\}$ ;            $\triangleright$  candidate band
13:    same as lines 10-13 of Algorithm 5;

```

---

## 6. POSTINGS MATERIALIZATION

Typical document archives cover a long period (e.g., 10 years), while user queries may apply to a relatively short period only (e.g., June 2005). If we use a single inverted list for each keyword, we might have to scan a large number of irrelevant postings to the query interval. To minimize the number of redundant accesses, in this section, we propose a specially designed R-tree for materializing the postings in each inverted list, which outperforms other

typical indexing techniques as shown in our experiments. In addition, we propose a technique that decomposes a durable top- $k$  query into multiple simple top- $k$  queries and a simpler durable top- $k$  query. This decomposition can further improve performance.

#### 6.1 Transformed R-tree

There are different possible approaches for organizing the contents of an inverted list in order to minimize the access of postings which are irrelevant to the query interval, and still allow access in decreasing score order for the relevant ones (to facilitate top- $k$  aggregation). These approaches include adaptations of interval indexes, such as the interval tree [9], the segment tree [9], and the R-tree [12], and data duplication with multiple inverted lists [5].

In this section, we propose an adaptation of the R-tree for this indexing problem. In Section 7, we compare our proposal to alternative indexes, including the proposal of [5]. The R-tree is a classic spatial access method, which divides the space with hierarchically nested, possibly overlapping, minimum bounding rectangles (MBRs). Figure 10(a) shows an example to build a R-tree index for an inverted list containing time-relevant document versions. We have 4 leaf nodes ( $m_1 - m_4$ ), 2 intermediate nodes ( $M_1, M_2$ ), and a root node ( $M$ ) in this tree and each MBR looks like a flat rectangle. Note that it is hard to avoid the high degree of overlapping in the R-tree when we have line segments in a 2D space (time-score). Because of the overlapping, there is higher chance to have *false hits*<sup>3</sup>, which degrade performance. For instance, assume that we have a query, as shown in the shaded area of Figure 2(a). This query intersects all leaf MBRs of the tree, meaning that all postings have to be examined in this example, although only 60% of them are actually relevant to the query interval.

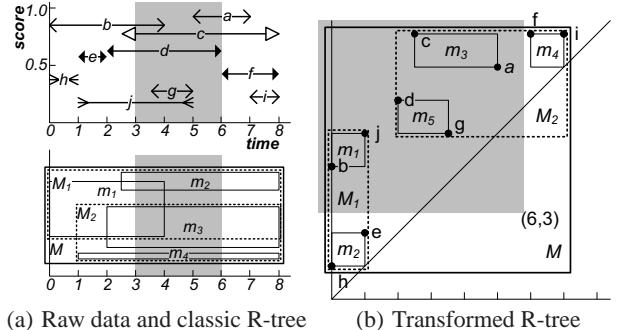


Figure 10: Different R-tree representations

Since grouping line segments is not effective for time-travel queries, we transform our data into a begin/endpoint space that supports better grouping. Now, point is the basic indexed unit, making grouping of data to nodes more effective. For each posting, we use a point  $(t_b, t_e)$  to represent it in a 2D space. We will explain how to handle the score ordering shortly. Figure 10(b) shows an example after the transformation. For instance, posting  $a$  with interval  $[5, 7]$  is represented by point  $(5, 7)$ . After the transformation, our query result is located in the shaded area. Although the shaded area is larger than the one in Figure 10(a), it intersects only 6 nodes ( $M, M_1, M_2, m_1, m_3$ , and  $m_5$ ) and there are fewer false hits.

To facilitate access of the postings that intersect the query interval in decreasing score order (as required by our aggregation algorithm), we pre-compute an aggregate score  $s_{max}$  for each MBR, where  $s_{max}$  stores the maximum score for all child MBRs. In addition, if the MBR is a leaf node,  $s_{max}$  stores the maximum score

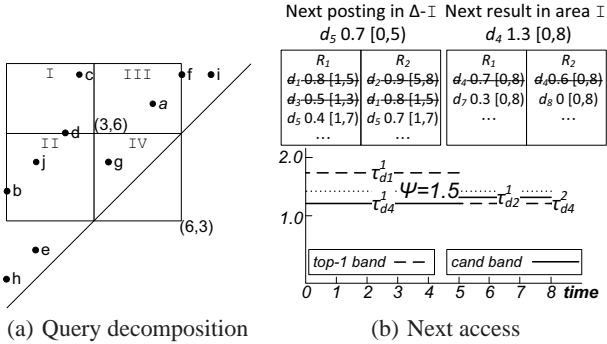
<sup>3</sup>a *false hit* is an accessed posting that does not intersect the query interval

of the postings inside. This scheme supports prioritized access of the MBRs that intersect the query interval  $q.\lambda$  by decreasing order of their aggregate scores. Starting from the root of the tree, each entry that intersects  $q.\lambda$  is inserted to a priority queue. The entry with the highest score is deheaped and the process is repeated for its children. When a posting (i.e., leaf node entry) is deheaped, we know that this corresponds to the next posting that intersects  $q.\lambda$  and has the highest score among all remaining such postings. Thus, the transformed R-tree elegantly combines temporal search based on  $q.\lambda$  and decreasing-score access order of the results.

## 6.2 A Partitioning-Based Approach

In this section, we investigate a query decomposition technique that further improves the performance of the band approach using the transformed R-tree. This technique aims at reducing the number of band maintenance operations. Recall that the exact score of a document version  $d_i$  at time  $t$  is unknown until it has been seen  $|L|$  times from the inverted lists, where  $|L|$  is the number of keywords in the query. Assume that  $\tau^i$  is the unit posting of  $d_i$  at timestamp  $t$ . For each new access of  $d_i$  at time  $t$ ,  $\tau^i$  is first removed from existing band structures (e.g., top bands/candidate band/candidate container) and then reinserted into the band structure with an updated score. Suppose that there is a method to determine the exact score of  $\tau^i$  before the first insertion; then  $\tau^i$  is processed only one time instead of  $|L|$  times. Based on this idea, we could further improve the performance of the proposed algorithms.

Looking at the postings distribution in the transformed R-tree, we observe that the order of some postings can be computed easily using a simple NRA query. Figure 11(a) shows an example that decomposes the space, based on query interval  $[3, 6)$ , into four areas: I, II, III, and VI. Note that area I contains all postings that fully cover the entire query interval, which are  $c$  and  $d$  in our example.



**Figure 11: Example of partitioning-based approach**

Our partitioning-based approach excludes area I from the durable query; we only issue a constrained NRA query to compute the top- $k$  result in this area. The results of this query should be merged with the results of the band approach in the remaining space. A trivial way to do this is to compute the exact top- $k$  result of the constrained NRA, and then merge it with the *partial* durable top- $k$  search in area  $\Delta - \text{I}$ , where  $\Delta$  represents the entire query area. While this approach guarantees correctness, we may have poor performance as more postings may be accessed compared to a single durable query in the whole area. A better approach is to integrate the constrained NRA query with our band maintenance algorithm. In order to support such a partitioning-based approach, we revise Algorithm 6, based on the following lines:

### New accessing approach :

Instead of reading the next posting from the inverted lists

(as in line 1 of Algorithm 6), it is read either from (i) the transformed R-tree excluding area I or (ii) the next result of the constrained NRA (using incremental search).

### Choosing the maximum last seen score :

Note that we have two sets of last seen scores. For each keyword, the maximum last seen score is chosen from area  $\Delta - \text{I}$  or area I.

### Revised termination condition :

Corollary 1 is an extended version of Lemma 1.

**COROLLARY 1.** *The top- $k$  result at timestamp  $t$  is stored in the top bands if and only if the score of the  $k$ -th band at time  $t$  is not worse than items (1) and (2) in Lemma 1, and (3) the next result from the constrained NRA.*

With the above modifications, we can access the posting arbitrarily from (i) or (ii) without affecting the correctness. In order to minimize the number of accesses, in our implementation, we define this order based on the best score (e.g.,  $\Psi$ ) of the partial durable query and the next element score in the constrained NRA.

We enrich the example in Figure 8(d) with more data to illustrate the partitioning-based approach. Note that the last seen score of  $L_1$  ( $L_2$ ) is  $0.7 = \max\{0.5, 0.7\}$  ( $0.8 = \max\{0.8, 0.6\}$ ). Currently,  $\Psi$  is set to  $1.5 (= 0.7 + 0.8)$ . According to the information in Figure 11(b), we know that the next posting is  $d_4$  with score 1.3 and interval  $[0, 8]$ , which is computed by the constrained NRA. When we insert this posting into the top bands, it is split into two unit postings  $\{\tau_{d_4}^1, \tau_{d_4}^2\}$  with intervals  $[0, 5)$  and  $[5, 8)$  respectively.  $\tau_{d_4}^1$  fails to enter the top-1 band but  $\tau_{d_4}^2$  succeeds to replace  $\tau_{d_2}^1$  and enters the top-1 band. Therefore, subroutine *insertBand* returns  $\{\tau_{d_4}^1, \tau_{d_2}^1\}$ . In the next loop, the constrained NRA is called to find the next top posting in area I. Suppose that the constrained NRA only reads one posting from each index and the next posting is  $d_7$  in area I. The last seen score of  $L_1$  ( $L_2$ ) becomes  $0.5 = \max\{0.5, 0.3\}$  ( $0.8 = \max\{0.8, 0\}$ ) and  $\Psi$  is updated to 1.3. After this update, we can terminate the search since now our top band is not worse than (1)  $\Psi$ , (2) the candidate band, and (3) the next result of the constrained NRA (see Corollary 1). Note that  $d_4$  is inserted only once into the top bands while it would be inserted twice using the original band approach.

The partitioning-based approach can be extended to further reduce the area where the band approach is applied. Note that area II (III) contains all postings that intersect  $q.t_b$  ( $q.t_e$ ) but not intersect  $q.t_e$  ( $q.t_b$ ). Based on our observation for area I, we can also add two constrained NRA in areas II and III. Finally, we decompose the original durable query into three constrained NRA queries plus one partial durable query only in area IV.

## 7. EXPERIMENTS

In this section we empirically evaluate the performance of our algorithms on the Wikipedia revision history, which is freely available at [www.wikipedia.org](http://www.wikipedia.org). The total size of the dataset used in our experiments is 0.7 TBytes, containing the full editing history from January 2001 to December 2005 of the entire English Wikipedia. The compression technique proposed in [5] is used to group similar consecutive versions of the same document, reducing the total size of the data to 0.15 TBytes. The resulting dataset contains a total of 892,255 documents (i.e., topics) with 13,976,915 versions, so there is a mean of 15.67 versions per document and a standard deviation of 59.18. Okapi BM25 [20] is used to normalize the term frequency with the length normalization parameter  $b = 1.2$  and the

tf-saturation parameter  $k_1 = 0.75$ . Inverted lists store postings of the form [doc-id, begin-time, end-time, score].

We selected the most frequent keyword queries (of 2 to 5 keywords) from a search engine log that yield a Wikipedia article as a web-search result. This guarantees that all keywords are relevant to Wikipedia articles. We classify a query  $q$  based on the total number of postings  $V(q)$  in the inverted lists of its keywords  $q.W$  and the correlation between the keywords. The correlation is defined by

$$R(q) = \frac{|\cap_{w_i \in q.W} D(w_i)|}{|\cup_{w_i \in q.W} D(w_i)|},$$

where  $D(w_i)$  denotes the set of documents containing keyword  $w_i$ . For instance, if a query  $q$  has  $V(q)$  and low  $R(q)$ , it is classified as **high volume and low correlation** ('HL' class in short). Accordingly, we have 4 classes in total, 'HH', 'HL', 'LH', and 'LL'. Some statistics for these classes are shown in Table 2: average number of postings per keyword, average interval length of postings, average number of distinct documents in postings of a keyword, and number of queries  $|Q|$  in class. The space of an inverted list (uncompressed) can be derived by multiplying the number of postings with the posting size (16 bytes). In addition, the average number of postings in class 'HH' is 64K, 202K, 713K, and 1.98M in years 2001, 2002, 2003, and 2004 respectively: more versions are created in more recent years.

In the experiments, we evaluate the scalability of our algorithms, including DAA (Section 4), BBA (Section 5.2), OBBA (Section 5.3), and the partitioning-based approach (PBA) (Section 6.2). We use LARA [18] (an optimized implementation of NRA) for NRA computations in DAA and PBA. Unless otherwise specified, in all experiments we selected queries from the 'HH' class.<sup>4</sup> In each experimental instance, 5 queries from the chosen class are used and the results are averaged.

**Table 2: Statistics of test queries in the four classes**

class	avg. postings per $w$	$ Q $	avg. length	avg. doc
HH	2.95M	61	45.79 days	41370.34
HL	3.32M	42	53.12 days	44873.87
LH	0.92M	39	36.45 days	13087.85
LL	0.77M	58	46.35 days	17117.49

All methods were implemented in C++ and the experiments were performed on an Intel Core2Duo 2.66GHz CPU machine with 4 GBytes memory, running on Ubuntu 8.04. Table 3 shows the ranges of the investigated parameters, and their default values (in bold). In each experiment, we vary a single parameter while setting the remaining ones to their default values.

**Table 3: Ranges of parameter values**

Parameter	Values
Number of keywords $ W $	2, 3, 4, 5
$k$	2, 5, <b>10</b> , 20, 40
Query length, $\lambda$ (in days)	15, 30, <b>60</b> , 120, 240
Query begin time, $t_b$ (in year)	2001, 2002, <b>2003</b> , 2004
Query class	<b>HH</b> , HL, LH, LL

## 7.1 Difference to Other Queries

First, we study how different the results produced by the durable top- $k$  query are, compared to simpler aggregation models. Table 4 shows the percentage of the durable top- $k$  results that are not generated by other aggregate queries. For example, DUR-MIN denotes

<sup>4</sup>typical searches include correlated keywords; we used keywords with a large number of postings to evaluate scalability.

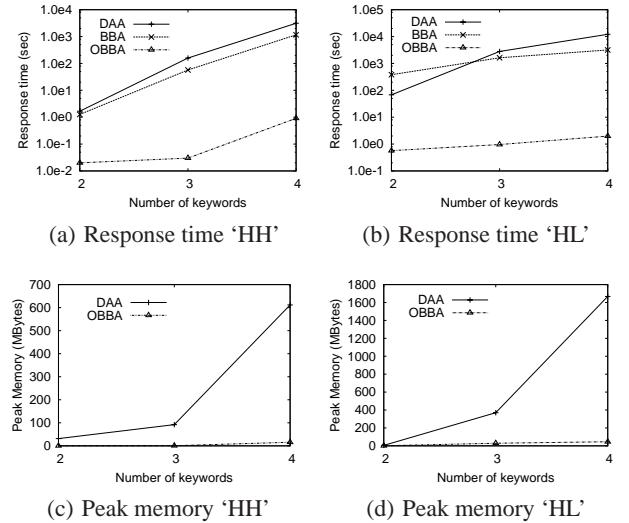
**Table 4: Result diversity in different queries**

$ W $	2	3	4	5	2	3	4	5		
$\lambda$	$\lambda = 60$ days					$\lambda = 120$ days				
DUR-MIN	10%	24%	6%	4%	26%	32%	12%	14%		
DUR-MAX	14%	20%	10%	4%	20%	14%	24%	28%		
DUR-AVG	34%	44%	10%	40%	36%	58%	16%	34%		

the set difference between the results of the durable and the MIN aggregate query. The queries are selected from class 'HH' (we found similar results when using other query classes) and we tested two query interval values ( $\lambda$ ). The query length  $|W|$  varies from 2 to 5 keywords. There is a significant difference in the durable top- $k$  results, compared to other models and the difference increases with  $\lambda$ , as larger intervals enclose more document versions. This shows that the durable query provides different and potentially more interesting results than simpler aggregation models.

## 7.2 Efficiency and Scalability

We now compare the durable top- $k$  algorithms in terms of efficiency and scalability. Figure 12 shows the response time and peak memory usage with respect to the number of keywords  $|W|$ , when keyword queries are selected from two classes: 'HH' and 'HL'. The optimized band approach (OBBA) always outperforms the other two methods, being 1-2 order of magnitudes faster in most cases. All methods become more expensive when there are more keywords in the query. This is consistent to the observations in [18]. All methods perform better for queries in class 'HH' than queries in class 'HL', since the correlation between keywords in class 'HH' is high; document versions of high scores in all keywords are found faster, assisting early termination of search.



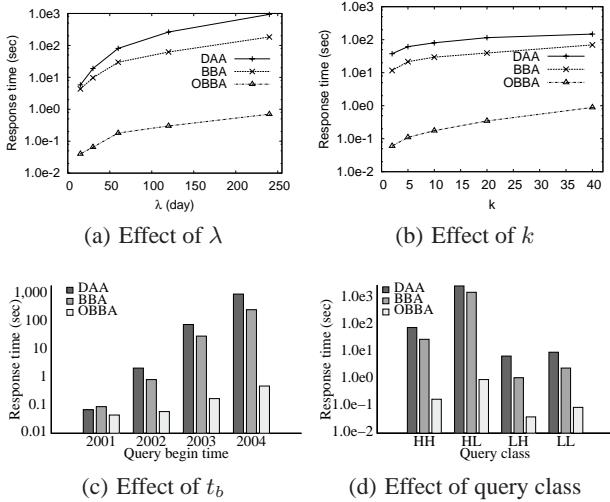
**Figure 12: Effect of  $|W|$**

Note that we skipped the case  $|W| = 5$  in Fig. 12. The reason is that DAA in this case consumes the physical memory of our system. In Figures 12(c) and 12(d), we show the peak memory usage of the methods during the query execution. DAA is more sensitive to the number of keywords in the queries than the band approach. The reason is that it runs  $O(m)$  NRA top- $k$  queries simultaneously, where  $m$  is equal to the number of postings that have been read. Each query consumes  $O(m)$  space in the worst case [18], therefore the space requirements of DAA are huge. On the other hand, the band approach stores only  $k$  top bands, one candidate band, and one

candidate container. The worst case complexity is  $O(km)$  which is much smaller than  $O(m^2)$  typically. As shown in Figure 12(d), DAA uses more than 1.6 GBytes memory. BBA is not included in this comparison, as it consumes similar memory to OBBA.

The next experiment studies the effect of different parameters in queries of three keywords, all taken from class ‘HH’. Figure 13(a) shows the response time of the methods as a function of the query length  $\lambda$ . We use two months as our default query length. When  $\lambda$  becomes larger, all methods become more expensive since we have more top- $k$  rankings while the query length becomes longer. Again, OBBA is 1-2 orders of magnitude faster than BBA and DAA. Moreover, OBBA accesses only 0.8%, 1.4%, and 2.4% of all intersected postings for queries in class ‘HH’ with query length  $\lambda$  60, 120, and 240 days respectively. This shows that our best method can compute the durable top- $k$  result by scanning only a small prefix of the inverted lists. Figure 13(b) plots the response time as a function of  $k$ . The effect is similar to  $\lambda$ ; the size of the top- $k$  ranked lists increase linearly with  $k$ , and the overhead of maintaining the lists/bands increases proportionally.

Figure 13(c) plots the response time of the methods as a function of query begin time  $t_b$ . Performance is sensitive to  $t_b$ , since more editors joined Wikipedia, creating more versions in the more recent years. For instance, topic “Ryan Giggs” was modified only 28 times in 2004, but it has been modified more than 572 times in this year up to November. As more versions enter the system, the problem itself becomes harder. The figure also demonstrates that our best method OBBA scales better than the other approaches. Figure 13(d) plots the response time of the methods as a function of all four query classes. As expected, when correlation increases all methods perform better, and when volume increases they become worse. OBBA is 1-3 orders of magnitude faster than BBA and DAA in all cases. In conclusion, OBBA consistently outperforms BBA and DAA by a wide margin at all tested cases and it has a low response time, making it practical in real scenarios.



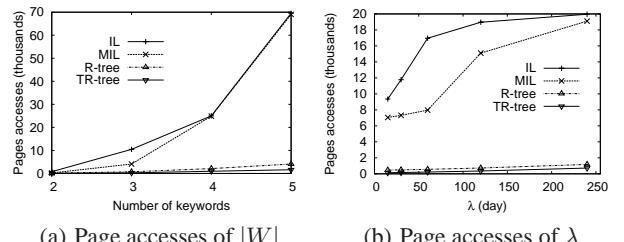
**Figure 13: Effect of different system parameters**

### 7.3 Postings Materialization and Partitioning

In the subsequent experiments, we test the effectiveness of the proposed storage and access scheme for the inverted lists (Section 6), comparing it with alternative approaches. We denote our transformed R-tree indexing scheme by TR-tree, and we include in the comparison (i) a single *inverted list* IL where postings are ordered by score only and  $\lambda$  is used for post-filtering them, (ii) the *multiple inverted lists* MIL approach of [5], and (iii) a simple R-tree, which

indexes the intervals instead of their transformation. The simple R-tree also stores aggregate score information at the MBRs and uses the same prioritized traversal as the TR-tree. We do not include the interval tree and segment tree in our experiments; according to our findings, they do not scale well since the versioned postings have a high overlap. Moreover, we split an IL into MIL using the method proposed by [5] by setting the space budget equal to the size of our TR-tree.<sup>5</sup> OBBA is used as the durable top- $k$  algorithm in all cases. Our system uses a 4Kb page size. In order to measure the exact I/O cost, we assume no memory buffer is available.

Figure 14(a) shows the page accesses of the methods as a function of the number of keywords. We set IL to be a baseline. MIL accesses fewer pages than IL but is at least 3 times worse than the TR-tree. The runner-up method R-tree is 2 times worse than the TR-tree. Moreover, the TR-tree is less sensitive to the number of keywords. Figure 14(b) shows the page accesses of the methods as a function of the query length  $\lambda$ . The trend is similar to Figure 14(a). The TR-tree accesses at least 3 times fewer pages than the runner up method. Note that MIL performs better when the query length fits in one/few small inverted list(s). Therefore, if the query length is small (15 days or smaller), MIL has higher chances to achieve good performance.

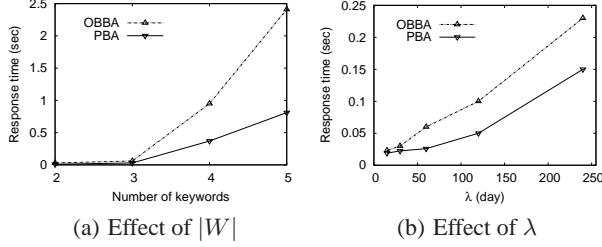


**Figure 14: Page accesses of different system parameters**

In the last experiment, we compare the direct use of the best durable top- $k$  algorithm, OBBA, with the partitioning-based approach (PBA) that decomposes a single durable query into multiple constrained NRA queries plus one partial durable query (Section 6.2). Figure 15 demonstrates the effectiveness of this strategy. As the number of keywords grows, PBA increasingly outperforms OBBA as shown in Figure 15(a). When  $|W| = 5$ , PBA is three times faster than OBBA. PBA maintains an advantage over OBBA also in the experiment of Figure 15(b), where the number of keywords is fixed to 3 and the query interval  $\lambda$  changes. We note that although PBA performs better than OBBA in terms of response time, it may access more pages from the TR-tree. The reason is that the same tree node may be accessed more than once by the constrained NRA queries and the partial durable query. In our experiments, PBA accesses around 10%-20% more I/O pages than OBBA. In practice, this does not affect the overall performance of PBA since these queries are clustered and the pages that are accessed more than once are already buffered.

In summary, the proposed TR-tree indexing scheme outperforms alternative approaches with respect to various parameters and greatly improves the I/O performance of OBBA. Moreover, it facilitates the application of the partitioning-based approach, which further reduces the response time of OBBA.

<sup>5</sup>The sizes of IL and the TR-tree are very similar, since point data are handled in a space-efficient way by the MBR structure. MIL also has similar size to the TR-tree due to the space budget setting. The simple R-tree occupies 25% more space than other methods since the postings highly overlap as shown in Figure 10(a).



**Figure 15: Effect of partitioning-based approach**

## 8. CONCLUSION

We studied the problem of durable top- $k$  search in document archives. We proposed two algorithms, the first is adapted from a typical solution; the second is based on a novel shared-execution idea. We tested our solutions on a large-scale corpus which includes all versions of Wikipedia pages from 2001 to 2005. Our experiments show that the fully optimized algorithm outperforms the simpler alternative by orders of magnitude in terms of response time, which typically is very low (tens of milliseconds). Our solution also includes an effective indexing approach for inverted lists, tailored for durable top- $k$  search and time-travel queries, in general. A space-partitioning approach can be applied with the help of this indexing scheme to decompose the durable query into three simple top- $k$  searches and a durable top- $k$  search in a constrained region. This approach greatly improves computational performance, especially for queries with multiple keywords.

As part of our future work, we plan adding special constraints to durable top- $k$  queries. For instance, an object is a *continuous durable top- $k$  result* if it is consistently in the top- $k$  results in a continuous time subinterval. Moreover, we plan to devise solutions that take advantage of the ratio  $r$  to accelerate search by pruning the space earlier. In addition, we will study the behavior of our proposed algorithm on other types of data, such as financial data or blogs. Finally, we will consider applying ideas from IR techniques, such as dividing inverted lists into segments based on scores [26], to further improve the efficiency of OBBA.

## Acknowledgments

This work was partially sponsored by Grant HKU 715509E from Hong Kong RGC. We would like Gerhard Weikum for his fruitful suggestions and Reza Sherkat for providing and preprocessing the Internet Archive data.

## 9. REFERENCES

- [1] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *SIGIR*, pages 372–379, 2006.
- [2] V. N. Anh and A. Moffat. Pruning strategies for mixed-mode querying. In *CIKM*, pages 190–197, 2006.
- [3] N. Bansal, F. Chiang, N. Koudas, and F. W. Tompa. Seeking stable clusters in the blogosphere. In *VLDB*, pages 806–817, 2007.
- [4] N. Bansal and N. Koudas. Blogscope: a system for online analysis of high volume text streams. In *VLDB*, pages 1410–1413, 2007.
- [5] K. Berberich, S. J. Bedathur, T. Neumann, and G. Weikum. A time machine for text search. In *SIGIR*, pages 519–526, 2007.
- [6] K. Berberich, S. J. Bedathur, and G. Weikum. Efficient time-travel on versioned text collections. In *BTW*, pages 44–63, 2007.
- [7] A. Z. Broder, N. Eiron, M. Fontoura, M. Herscovici, R. Lempel, J. McPherson, R. Qi, and E. J. Shekita. Indexing shared content in information retrieval systems. In *EDBT*, pages 313–330, 2006.
- [8] G. M. D. Corso, A. Gulli, and F. Romani. Ranking a stream of news. In *WWW*, pages 97–106, New York, NY, USA, 2005. ACM Press.
- [9] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 3rd ed. edition, 2008.
- [10] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing tags over time. *ACM Trans. Web*, 1(2):7, 2007.
- [11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.
- [12] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.
- [13] J. He, H. Yan, and T. Suel. Compact full-text indexing of versioned document collections. In *CIKM*, pages 415–424, 2009.
- [14] M. Herscovici, R. Lempel, and S. Yogev. Efficient indexing of versioned document sequences. In *ECIR*, pages 76–87, 2007.
- [15] J. Kleinberg. Temporal Dynamics of On-Line Information Streams. In M. Garofalakis, J. Gehrke, and R. Rastogi, editors, *Data Stream Management Processing High-Speed Data Streams*. Springer-Verlag, 2006.
- [16] M.-L. Lee, W. Hsu, L. Li, and W. H. Tok. Consistent top- $k$  queries over time. In *DASFAA*, pages 51–65, 2009.
- [17] X. Li and W. B. Croft. Time-based language models. In *CIKM*, pages 469–475, New York, NY, USA, 2003. ACM.
- [18] N. Mamoulis, M. L. Yiu, K. H. Cheng, and D. W. Cheung. Efficient top- $k$  aggregation of ranked inputs. *ACM Trans. Database Syst.*, 32(3):19, 2007.
- [19] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR*, pages 275–281, New York, NY, USA, 1998. ACM.
- [20] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR*, pages 232–241, New York, NY, USA, 1994. Springer-Verlag New York, Inc.
- [21] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In *SIGMOD Conference*, pages 76–85, 2003.
- [22] R. Sherkat and D. Rafiei. On efficiently searching trajectories and archival data for historical similarities. *PVLDB*, 1(1):896–908, 2008.
- [23] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *SIGIR*, pages 21–29, 1996.
- [24] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.
- [25] J. Zhang and T. Suel. Efficient search in large textual collections with redundancy. In *WWW*, pages 411–420, New York, NY, USA, 2007. ACM.
- [26] M. Zhu, S. Shi, M. Li, and J.-R. Wen. Effective top- $k$  computation with term-proximity support. *Inf. Process. Manage.*, 45(4):401–412, 2009.