

Efficient Temporal Keyword
Queries
over Versioned Text

Avishek Anand
Srikanta Bedathur
Klaus Berberich
Ralf Schenkel

MPI-I-2010-5-003 October 2010

Authors' Addresses

Avishek Anand
Max-Planck-Institut für Informatik
66123 Saarbrücken
Germany

Srikanta Bedathur
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

Klaus Berberich
Max-Planck-Institut für Informatik
66123 Saarbrücken
Germany

Ralf Schenkel
Saarland University
66123 Saarbrücken
Germany

Abstract

Modern text analytics applications operate on large volumes of temporal text data such as Web archives, newspaper archives, blogs, wikis, and micro-blogs. In these settings, searching and mining needs to use constraints on the time dimension in addition to keyword constraints. A natural approach to address such queries is an inverted index whose entries are enriched with valid-time intervals. It has been shown that these indexes have to be partitioned along time in order to achieve efficiency. However, when the temporal predicate corresponds to a long time range which overlaps with multiple partitions, naive query processing incurs high cost of reading of redundant entries across partitions.

We present a framework for efficient approximate processing of keyword queries over a temporally partitioned inverted index which minimizes this overhead, thus speeding up query processing. By using a small synopsis for each partition we identify partitions that maximize the number of final non-redundant results, and schedule them for processing early on. Our approach aims to balance the estimated gains in the final result recall against the cost of index reading required. We present practical algorithms for the resulting optimization problem of index partition selection. Our experiments with 3 diverse, large-scale text archives reveal that our proposed approach can provide close to 80% result recall even when only about half the index is allowed to be read.

Keywords

Temporal Text Indexing, Time-Travel Text Search, Web Archives

Contents

1	Introduction	3
1.1	Motivation	3
1.2	Selecting Partitions for Query Processing	4
1.3	Contributions	6
2	Preliminaries	7
2.1	Model	7
2.2	KMV Synopses	8
3	Single-term Partition Selection	10
3.1	Size-based Partition Selection	10
3.2	Equi-cost Partition Selection	12
3.3	Approximation Algorithm	13
4	Multi-term Partition Selection	16
4.1	Solution	17
5	Practical Issues	21
6	Evaluation Framework	23
6.1	Setup	23
6.2	Datasets and Query Workload	23
6.3	Index Management	24
6.4	Evaluation Methodology	26
7	Experimental Results	27
7.1	Performance of Partition Selection	27
7.2	Query Runtimes	28
7.3	Impact of using Synopses	29
7.4	Impact of Partition Granularity	30
8	Related Work	35

1 Introduction

1.1 Motivation

Large-scale versioned text data is increasingly becoming abundant in the form of archives of the Web, corporate/CRM records, wikis, blogs, micro-blogs, etc. The history of information evolution buried in these collections is an important source of actionable intelligence in a variety of applications. It is often necessary to retrieve documents from these collections that satisfy certain content predicates, expressed typically through keyword queries, as well as temporal predicates, e.g., on the time when they were published or accessible on a Web server. As a concrete example, consider a business analyst looking for web pages predicting and analyzing upcoming releases of tablet computers. If only keyword queries are used to retrieve pages from a Web archive, irrelevant information about earlier releases of tablet computers may corrupt the analytics results. By including a temporal constraint on the publication or discovery time of web pages, such undesirable results can be eliminated. Queries that combine the content and temporal predicates are termed as *time-travel queries*.

Processing time-travel queries is much more expensive than processing plain keyword queries without temporal constraints. Using standard preprocessing techniques from information retrieval, a naive implementation could build inverted lists that store, for each term, all documents that contain this term, enriched by their lifespan. However, when processing time-travel queries, a large fraction of reads from the inverted lists would be wasteful – i.e., do not contribute to the final result – namely all the entries that do not qualify for the temporal predicate of the query. In [7] these issues were addressed for the restricted class of time-travel queries referring to a single point of time in the past. That approach partitioned inverted lists along time, resulting in list partitions that contain all entries whose lifespan overlaps with the time interval assigned to the partition. For processing a time-point query in the resulting temporally partitioned index, it is sufficient to consider just

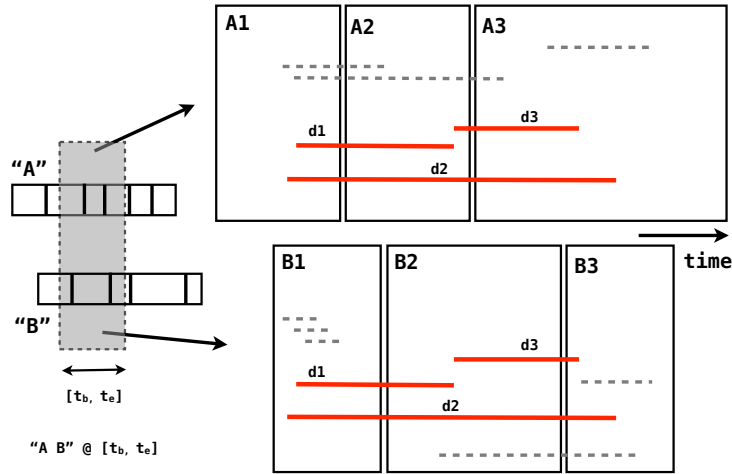


Figure 1.1: Processing a time-travel query “A B” @ $[t_b, t_e]$ using partition selection

one partition for each term, leading to significant reduction in I/O and computational costs during query processing. As a natural side-effect of this temporal partitioning, documents with long lifespans are replicated across several adjacent partitions.

For the much more practical and general class of time-travel queries where the temporal predicate is a *time range*, the straight-forward query processing of [7] quickly becomes inefficient as a consequence of this replication, as repeatedly reading replicated entries from several partitions within the query time range wastes I/O operations. The simple alternative of not partitioning is also not desirable, as it penalizes queries with shorter time-range or time-point predicates by having to scan complete lists. The effects of this are further exacerbated in the typical text analytics process involving multiple interactive steps of query reformulation, expansion, and refinement, which require quick turnaround times.

1.2 Selecting Partitions for Query Processing

In analytical settings like those outlined before, a user does not necessarily require the exact query result, but often would be satisfied with a good approximation that is determined quickly. In many application domains like news articles, the same information is available from different documents, so missing a few of them could be acceptable. Similarly, a subset of the true results is usually good enough for quickly checking if content or temporal predicates of the query need to be adapted.

This paper introduces an approach for approximate processing of time-travel queries in versioned text collections that allows to trade in some result quality for improved retrieval time. It exploits the following observation: if we can determine that a partition largely consists of entries that are replicated in already processed partition(s), we can avoid processing this partition without significantly compromising the final result quality. We aim at *selecting a set of partitions* which can be processed within a given maximal processing cost and that yield a good approximate query result. We consider abstract cost measures for processing a query, namely the number of partitions touched or the number of index entries read during execution. A user would specify bounds on the execution time that can be transformed into bounds on the abstract cost by the system. Alternatively, the user can also stop the processing at any time when she determines that the results are already satisfying (or the query needs to be refined); our greedy methods support this by selecting partitions first that are likely to contain many unseen answers.

We use the toy example in Figure 1.1 to illustrate the general idea of our approach. This figure shows two inverted lists for two terms A and B built over documents with lifetimes. On the left side of the figure, individual inverted lists are shown, each spanning the entire time interval. The gray shaded region represents a temporal predicate that spans a small time range over these lists. In the absence of temporal partitioning, query processing needs to entirely scan both lists entirely and filter out entries that do not satisfy the temporal predicate. When the index is partitioned, however, the processing can be speeded by reading only the relevant partitions that overlap with the temporal predicate, represented on the right side of the figure. Thus, in our example, a total of 6 partitions – A_1, A_2, A_3 and B_1, B_2, B_3 – have to be processed to determine the answer set of $\{d_1, d_2, d_3\}$ – marked as red line-segments.

However, a closer inspection of Figure 1.1 reveals that the same answer set can be obtained by processing only 2 *partitions*, A_2 and B_2 , since the replicas of index entries for documents in the answer set are fully available within these two partitions.

How can we make use of this observation in practice? In order to do so, we need to answer the following questions: (i) does a partition being considered contribute non-redundantly towards the final answer set when it is processed, and by how much? (ii) is there an alternate set of partition(s) which can contribute these answers at a lower access cost? Based on answers to these questions, we can generate a partition access plan so that for a specified cost budget only those partitions are chosen for processing which maximize the number of results.

1.3 Contributions

In this paper, we address these issues in detail and propose a query processing approach which places the ability to directly control the performance in the hands of the analyst posing the query. We consider the following problem:

Problem 1 *Given an upper-bound on the I/O cost incurred during query processing, select a subset of partitions so that the recall of the final result is maximized without exceeding the specified upper-bound.*

We formally model these *partition selection* problems as optimization problems, distinguishing two alternative formulations: a) *Size-based Partition Selection* – where the I/O cost of accessing a partition is proportional to the size of the partition, and b) *Equi-cost Partition Selection* – where the latency of simply accessing a partition is predominant, irrespective of the size of the partition.

Making use of KMV synopses for cardinality estimates under set operations [8], we develop algorithms for efficiently solving such partition selection problems. In particular, the contributions of this paper are:

1. A optimal dynamic programming based algorithm partition selection for queries with single keyword,
2. An efficient greedy alternative for partition selection that can be applied for both single keyword as well as multi-keyword queries,
3. A detailed experimental evaluation on 3 large-scale real-world text archives: Wikipedia version history, a Web archive, and the Annotated New York Times archive spanning 20 years.

Our experimental evaluation shows that our proposed methods can compute more than 80% of final results even when the I/O budget is set as low as 50% of the total size of the partitions that satisfy the temporal predicate.

2 Preliminaries

2.1 Model

We operate on a document collection \mathcal{D} . Each document $d \in \mathcal{D}$ from the collection has a unique identifier id_d and consists of terms drawn from a vocabulary \mathcal{V} , i.e., $d \subseteq \mathcal{V}$. Furthermore, each document has an associated valid-time interval $[b_d, e_d)$ that conveys when the document existed in the real world. For simplicity of presentation, we assume that each document exists in exactly one version, and we will talk only about documents from now on. This restriction can be easily lifted by identifying a version of a document by the identifier and the begin timestamp of the version, and returning every version satisfying the temporal predicate and containing the queried text as a result.

We assume that documents are indexed on a per-term basis, in the spirit of an inverted index, and that the index is temporally partitioned. In detail, we let \mathcal{P}_v denote the set of partitions of the inverted list L_v for term $v \in \mathcal{V}$. Every partition $\mathcal{P}_{v,j}$ has an associated time interval $[b_{v,j}, e_{v,j})$ and contains (identifiers of) all documents in L_v that existed at any time during the time interval associated with \mathcal{P}_v , i.e.,

$$\mathcal{P}_{v,j} = \{ d \in \mathcal{D} \mid v \in d \wedge [b_d, e_d) \cap [b_{v,j}, e_{v,j}) \neq \emptyset \} .$$

Further, for the scope of this work, we assume that time intervals associated with partitions for term v are disjoint, i.e.,

$$\forall i \forall j : [b_{v,i}, e_{v,i}) \cap [b_{v,j}, e_{v,j}) = \emptyset .$$

Such temporally partitioned inverted lists have first been proposed for the Time-Travel Index (TTIX) in [7]. It maintains, for each partition, a list with entries for all documents in that partition that are augmented by validity-time intervals, which are thus of the form $\langle id_d, b_d, e_d, tf \rangle$ where id_d is a document identifier, $[b_d, e_d)$ is the validity-time interval of that document,

and tf is the frequency of the list’s term in the document. The partitioning strategies introduced in [7] trade-off extra storage-costs and query-processing gains.

A time-travel query, as considered in this work, consists of a set of terms $Q = \{q_1, \dots, q_m\}$ and a time interval $[b_q, e_q]$. The result of the query $Q@[b_q, e_q]$ is defined as the set of documents that contain all terms from Q and existed at any time during the time interval $[b_q, e_q]$, that is formally:

$$R(Q@[b_q, e_q]) = \{d \in \mathcal{D} \mid \forall q \in Q : q \in d \wedge [b_d, e_d] \cap [b_q, e_q] \neq \emptyset\} .$$

Queries for which $b_q = e_q$ holds, so that the query time-interval collapses into a single time point, will be referred to as *time-point queries*.

2.2 KMV Synopses

In many stages of our proposed approach, we depend critically on obtaining high quality cardinality estimates under union and intersection of large sets of document ids. For this purpose, we utilize recently proposed KMV synopses [8]. In a precomputation step, we build and store on disk the synopses for each partition of the temporally partitioned index, which we use during our partition selection process. In this section, we provide a brief background on KMV synopses.

We need to obtain good cardinality estimates Beyer et al. [8] introduced *KMV synopses* as effective sketches for sets that support arbitrary multiset operations including union, intersection, and differences. A KMV synopsis for a multiset S is created as follows: Fix a hash function h of the form $h : \Theta(S) \mapsto 0, 1, \dots, M$ where $\Theta(S)$ contains the distinct values in S and $M = O(|\Theta(S)|^2)$. The hash function h is applied to value of $\Theta(S)$, and the k smallest of the hashed values form the KMV (for *k minimum values*) synopsis L_S of S .

KMV synopses can deal with a variety of multiset operations (union, intersection, difference). The following equation computes an unbiased estimate for $|\Theta(S)|$, the number of distinct values in S , from the KMV synopsis, where U_k is the value of the k ’th smallest value:

$$\hat{D}_k = (k - 1)/U_k \tag{2.1}$$

Given two multisets A and B with their KMV synopses L_A and L_B of size k_A and k_B , respectively, it is possible to estimate the number of distinct values in the union of A and B as $D_\cup = |\Theta(A \cup_m B)|$ (where \cup_m denotes the

union of two multisets). Let $L = L_A \oplus L_B$ be defined as the set including the k smallest values in $L_A \cup L_B$, where $k = \min(k_A, k_B)$ and L is the KMV synopsis of size k describing $L_A \cup_m L_B$. D_\cup is estimated by the following equation:

$$\hat{D}_\cup = (k - 1)/U_k \tag{2.2}$$

A similar estimator can be developed for $D_\cap = |\Theta(A \cap_m B)|$, the number of distinct values in the multiset intersection of A and B ; we omit details for space reasons.

3 Single-term Partition Selection

We first focus on the special case where the time-travel keyword query $Q@[b_q, e_q]$ consists of only a single query term, i.e., $Q = \{q\}$. Our objective when selecting partitions to process the time-travel keyword query is to retrieve as many of the original query results as possible, while not violating a user-specified I/O bound. Our optimization criterion, to put it differently, is to maximize the *relative recall* as the fraction of original query results retrieved. The user-specified I/O bound, which constrains the space of valid solutions, can either be entry-based (*size-based partition selection*) or partition-based (*equi-cost partition selection*). In the former case, we are allowed to read up to a fixed number of index entries; in the latter case, we are allowed to select up to a fixed number of partitions.

3.1 Size-based Partition Selection

The input to this optimization problem is the set of affected partitions $\mathcal{P}_{q,j}$ with $1 \leq j \leq m$, and the user-specified I/O bound β as the fraction of entries of affected partitions that we are allowed to read. Formally, the problem can be stated as

$$\begin{aligned} \max & \left| \bigcup_{j, x_j \neq 0} \mathcal{P}_{q,j} \right| \quad \text{s.t.} \\ & \sum_j x_j \cdot |\mathcal{P}_{q,j}| \leq \beta \cdot \left(\sum_j |\mathcal{P}_{q,j}| \right) \\ & x_j \in \{0, 1\}. \end{aligned}$$

where x_j is an indicator variable which denotes whether the partition $\mathcal{P}_{q,j}$ is selected.

The above problem can be solved using dynamic programming over an increasing number of partitions affected. We build a dynamic programming table, DP , such that each cell $DP[c][p]$ represents the set of selected partitions for the prefix subproblem which considers the affected partitions $\{\mathcal{P}_{q,1} \cdots \mathcal{P}_{q,p}\}$, and the capacity is set to c . Theorem 1 proves that the recall for such a subproblem can be computed by reusing the solution of the subproblems $DP[c'][p']$ ($0 \leq c' \leq c$ for integral values of c').

Theorem 1. *Let $r(S_{c,k})$ denote the recall obtained by $S_{c,k}$, a subset of partitions selected from the ordered set, $\{\mathcal{P}_{q,1} \cdots \mathcal{P}_{q,k}\}$, and satisfying the capacity c . Let $\bar{S}_{c,k}$ be the selection of partitions such that the recall is maximized, i.e., $\bar{S}_{c,k} = \underset{S_{c,k}}{\operatorname{argmax}} r(S_{c,k})$. Then,*

$$r(S_{c,k}) = \max \left\{ r(\bar{S}_{c,k-1}), \max_{0 < k' < k} r(\bar{S}_{(c-|\mathcal{P}_{q,k}|),k'} \cup \{\mathcal{P}_{q,k}\}) \right\}$$

Proof. Assume that we have optimal solutions for all subproblems with capacities less than c , for the set of partitions $\{\mathcal{P}_{q,1}, \dots, \mathcal{P}_{q,i}\}$ where $0 < i < k$. Now we consider computing the optimal selection set $\bar{S}_{c,k}$ for the subproblem with a capacity c , and an ordered set of partitions $\{\mathcal{P}_{q,1} \cdots \mathcal{P}_{q,k}\}$. Let us denote the optimal recall value to be $OPT_{c,k}$ and assume that $OPT_{c,k} > r(\bar{S}_{c,k})$.

Case 1 – $\bar{S}_{c,k}$ does not include $\mathcal{P}_{q,k}$: This means $OPT_{c,k} > r(\bar{S}_{c,k-1})$, since $r(\bar{S}_{c,k}) = r(\bar{S}_{c,k-1})$ when $\mathcal{P}_{q,k}$ is not included in the $\bar{S}_{c,k}$. As a consequence, we have a new optimal solution for the subproblem for capacity c and the set of partitions $\{\mathcal{P}_{q,1}, \dots, \mathcal{P}_{q,k-1}\}$. But, this is contrary to our initial assumption that we already have the optimal recall values for the prefix subproblem. Thus, by contradiction or claim in the theorem holds.

Case 2 – $\bar{S}_{c,k}$ to obtain $OPT_{c,k}$ includes partition $\mathcal{P}_{q,k}$: This means $OPT_{c,k} > r(\bar{S}_{(c-|\mathcal{P}_{q,k}|),k'} \cup \{\mathcal{P}_{q,k}\})$. Let us denote the index of the partition selected just before $\mathcal{P}_{q,k}$ to be k' . Hence,

$$OPT_{c,k} - |\mathcal{P}_{q,k} \cap \mathcal{P}_{q,k'}| > r(\bar{S}_{(c-|\mathcal{P}_{q,k}|),k'})$$

where $k' = \underset{S}{\operatorname{argmax}} OPT \setminus \{\mathcal{P}_{q,k}\}$

This is contrary to our assumption and hence by contradiction our claim in the Theorem holds true. \square

Lemma 1, paves way for efficiently filling the dynamic programming table for the defined optimization problem.

Lemma 1. *For a set of partitions belonging to a term, the overlaps of contents of partition $\mathcal{P}_{q,i}$ with $\mathcal{P}_{q,i+k}$, $\forall k \geq 0$ have the following property:*

$$\mathcal{P}_{q,i} \cap \mathcal{P}_{q,i+k} \subseteq \mathcal{P}_{q,i+j} \mid 0 \leq j \leq k$$

Each of the DP table cell contains a pair of values – the last partition selected, lp , for the corresponding subproblem (i.e., the selected partition with the maximum begin-time), and the optimal recall value r . Since the choice of partitions cannot be made independently, the computation of recall for a newly selected partition takes into account only the entries that are not already included in previously selected partitions. Using Lemma 1, we can efficiently compute the optimal recall for each subproblem since all overlaps with the preceding partitions to lp are already covered in lp . The DP-based algorithm, outlined in Algorithm 1, has a time complexity of $\mathcal{O}(n^2 \cdot (\sum_j |\mathcal{P}_{q,j}|))$ and a space complexity of $\mathcal{O}(n \cdot (\sum_j |\mathcal{P}_{q,j}|))$.

The optimal partitioning can be easily computed by retracing the path taken by the best solution seen at $DP_{lp}[c_{max}][n]$.

3.2 Equi-cost Partition Selection

The inputs to this problem are the set of affected partitions $\mathcal{P}_{q,j}$, and bound β as a fraction of the number of affected partitions, N , to be read. We use the previous notations and formally state the problem as

$$\max \left| \bigcup_{j, x_j \neq 0} \mathcal{P}_{q,j} \right| \quad \text{s.t.}$$

$$\sum_j x_j \leq \beta \cdot N$$

$$x_j \in \{0, 1\}, \quad \forall \mathcal{P}_{q,j}.$$

It can be observed that this is a special case of the problem defined in Section 3.1 above, obtained by setting the value of 1 as the cost of each partition. However, by employing a uniform cost per partition, allows improvements in both the space and time complexity of the algorithm, as the optimal value is independent of the sum of the sizes of the partitions read. As a result, the time complexity of the algorithm reduces to be $\mathcal{O}(n^3)$ and a space complexity of $\mathcal{O}(n)$.

Algorithm 1 Partition Selection - Dynamic Programming solution

```
1:  $c_{max} = \lfloor \beta_e \cdot (\sum_j |\mathcal{P}_{q,j}|) \rfloor$ 
2:  $DP[0..c_{max}][0..n]$  // dynamic programming table
3:
4: for  $i = 0..c_{max}$  do
5:    $DP[i][0] = \emptyset$ 
6: end for
7:
8: for  $k = 1..n$  do
9:   for  $i = 0..|\mathcal{P}_{q,k}| - 1$  do
10:     $DP[i][k] = \emptyset$  // no partitioning possible
11:   end for
12:   for  $i = |\mathcal{P}_{q,k}|..c_{max}$  do
13:     for  $k' = 0..k - 1$  do
14:       //update if recall is better than current vale
15:        $r_{k'} = DP_r[i - |\mathcal{P}_{q,k}|][k'] + (|\mathcal{P}_{q,k}| - (|\mathcal{P}_{q,k} \cap DP_{lp}[i - |\mathcal{P}_{q,k}|][k'])))$ 
16:     end for
17:      $k' = \underset{r_{k'}}{argmax}$ 
18:     // Update the DP table with the best partitioning
19:      $DP_r[i][t] = max\{DP_r[c_j][k_i - 1], r_{k'}\}$ 
20:      $DP_{lp}[i][t] = \underset{DP_r[i][t]}{argmax}$ 
21:   end for
22: end for
23:
24: return  $DP[c_{max}][n]$ 
```

3.3 Approximation Algorithm

While the selection algorithms outlined above allow for polynomial run times, they are not efficient enough to be applied during query processing. Alternatively, we propose the use of $(1 - \frac{1}{e})$ -approximation algorithm called *Greedy-Select*, developed in [10] for solving *budgeted maximum coverage* (BMC) problem. We first show the equivalence of our partition selection problem and the BMC problem.

Definition 1 (Budgeted Maximum Coverage). *A collection of sets $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_m\}$ with associated costs $\{c_i\}$ is defined over a domain of elements $X = \{x_1, x_2, \dots, x_n\}$ with associated weights $\{w_i\}$. The goal is to find a collection of sets $\mathcal{S}' \subseteq \mathcal{S}$, such that the total cost of the elements in*

\mathcal{S}' does not exceed a given budget L , and the total weight of every element covered by \mathcal{S}' is maximized.

Theorem 2. *Partition selection is equivalent to budgeted maximum coverage.*

Proof. The selection problem for single terms can be cast into an instance of the Budgeted Maximum Coverage (BMC) problem [10] in the following way: The affected partitions, $\mathcal{P}_{q,j}$'s, are the analogous to the *sets* in the BMC problem with the entries in the partitions being the *elements* of the respective *set*. For size-based partition selection the cost for each set is its cardinality; for equi-cost selection, the cost for each set is unity. The cost budget is exactly the I/O bound IO_BOUND. With this reduction we can use the approximation algorithm proposed by Khuller et al. [10] which has a constant approximation guarantee of $(1 - \frac{1}{e})$. \square

The greedy approximate algorithm, *GreedySelect* is shown in Algorithm 2. In this algorithm, every partition is associated with a cost (c_i) and a benefit (B_i). Let the selection set be denoted as \bar{S} for simplicity. The cost of an unselected partition $\mathcal{P}_{q,j}$ ($\mathcal{P}_{q,j} \notin \bar{S}$) is the number of entries in the partition (for size-based selection) or 1 (for equi-cost selection). Its benefit, B_i , is the number of unread/uncovered entries in $\mathcal{P}_{q,j}$, i.e., $|\mathcal{P}_{q,j} \setminus \cup_{s \in \bar{S}} s|$. We additionally define IO_BOUND as the bound on the amount of I/O allowed (in terms of number of entries or number of partitions read). Each iteration of *GreedySelect* consists of a **selection step**, where the best partition in the current state is chosen and added to the selection set \bar{S} , followed by an **update step**, where the benefits of the remaining partitions are adjusted. *GreedySelect* chooses the best partition based on a greedy heuristic that picks at each iteration a partition that maximizes the benefit/cost ratio $\frac{B_i}{c_i}$ while adhering to the space constraint.

Algorithm 2 GreedySelect : Approximate Partition Selection

```
1:  $c_{max} = \lfloor IO\_BOUND \rfloor$ 
2:  $\bar{S} = \emptyset$ 
3:  $\mathcal{A} = \mathcal{P}_q$ 
4:  $C = 0$ 
5:
6: repeat
7:   Select  $\mathcal{P}_{q,i} \in \mathcal{A}$  that maximizes  $\frac{B_i}{c_i}$ 
8:   if  $C + c_i \leq c_{max}$  then
9:      $\bar{S} = \bar{S} \cup \mathcal{P}_{q,i}$ 
10:     $C = C + c_i$ 
11:   end if
12:    $\mathcal{A} = \mathcal{A} \setminus \mathcal{P}_{q,i}$ 
13: until  $\mathcal{A} = \emptyset$ 
14:
15: Select a partition  $\mathcal{P}_{q,t}$  that maximizes  $B_t$  over  $S$ 
16: if  $B(\bar{S}) \geq B_t$  then
17:   output  $\bar{S}$ 
18: else
19:   output  $\{\mathcal{P}_{q,t}\}$ 
20: end if
```

4 Multi-term Partition Selection

In the case of partition selection for single-term queries, every document entry read from a partition qualifies as an answer, given that the time-range of the partition overlaps with query time-range. Unlike this simpler setting, for multi-term queries there is an additional constraint imposed by the *conjunctive semantics* of query evaluation which requires that every result document also contain *all* the query keywords. Mimicking the conventional query processing (over standard inverted lists), multi-term queries can be evaluated by intersecting partitions of individual query terms. Now, the partition selection aims to increase the coverage of entries that belong to this *intersection space* of partitions.

We denote by SPS and EPS the size-based and equi-cost partition selection respectively, and define them formally for multi-term queries as follows:

$$\begin{aligned} & \max \left| \bigcap_{1 \leq i \leq m} \bigcup_{j, x_{ij} \neq 0} \mathcal{P}_{i,j} \right| \quad \text{s.t.} \\ & \sum_i \sum_j x_{ij} \cdot |\mathcal{P}_{i,j}| \leq \beta_e \cdot \left(\sum_i \sum_j |\mathcal{P}_{i,j}| \right) \end{aligned} \quad (\text{SPS})$$

or,

$$\sum_i \sum_j x_{ij} \leq \beta \cdot N, \quad (\text{EPS})$$

and,

$$x_{ij} \in \{0, 1\} .$$

where x_{ij} is an indicator variable which denotes whether a partition $\mathcal{P}_{i,j}$ is selected for processing.

4.1 Solution

The intersection space, in the objective function above, is the intersection of the unions of the affected partitions. Using the distributive property of the set intersection operator we can represent the above into unions of intersection ($\bigcap_{1 \leq j \leq m} \bigcup \mathcal{P}_{i,j} = \bigcup \bigcap_{1 \leq j \leq m} \mathcal{P}_{i,j}$). Let each of these resulting smaller intersections, consisting of one partition each from every term, be represented as a tuple \mathbf{x} . It is easy to see that these tuples come from the cartesian product among partition sets \mathcal{P}_q or for a m-term query $\mathcal{X} = \mathcal{P}_1 \times \dots \times \mathcal{P}_m$ (denoting the cartesian product set as \mathcal{X}). We formally define \mathbf{x} , an element of this cartesian product set \mathcal{X} , as :

$$\mathbf{x} = \{ (x_1, \dots, x_m) \mid x_i \in \mathcal{P}_i \}$$

We can now use *GreedySelect* over \mathcal{X} , where each element \mathbf{x} is equivalent to a partition in single-term selection scenario. The *benefit* of \mathbf{x} is defined as the cardinality of the documents in the intersection of the partitions in \mathbf{x} which are not in the selection set \bar{S} . In other words, the benefit or contribution of \mathbf{x} represents the number of *new documents* which are present in *every* element partition of \mathbf{x} . The *cost* definition of \mathbf{x} depends on the sizes of the element partitions in \mathbf{x} . Similar to our assumptions earlier Size-based selection sets the cost of \mathbf{x} as the sum of the sizes of the participating/element partitions not in \bar{S} . Equi-cost selection on the other hand defines the cost of \mathbf{x} as the number of participating partitions not in \bar{S} .

The reconstructed inputs to the algorithm *GreedySelect* is now the set \mathcal{X} , with defined benefits and cost for each of its elements, and the IO_BOUND derived from β or β_e depending on the variation of problem used. *GreedySelect* now proceeds conventionally by greedily choosing the \mathbf{x} with the best benefit by cost ratio. Observe that the choices of elements from \mathcal{X} are not independent. Selection of a certain element, \mathbf{x} , might result in reducing the cost (not the case in single-term selection) of others which have at least one of the constituent partitions common with \mathbf{x} . Hence in the updation step, apart from updating the benefit of \mathbf{x} , we also update its cost. Because of this varying costs characteristic the approximation-guarantee for the algorithm does not apply but is seen to work well in practice.

The cartesian set of partitions might be large, particularly if the number of terms in a query or the partitions per term or both are larger in number. In such a scenario *GreedySelect* can progressively get inefficient because of the numerous update cycles. To alleviate this we operate on a constrained set, τ -set $\subseteq \mathcal{X}$, which has a cardinality linear in the number of participating partitions as opposed to high number of combinations in \mathcal{X} . This constrained set is obtained by defining a τ -join operation over the term-partition sets \mathcal{P}_i 's

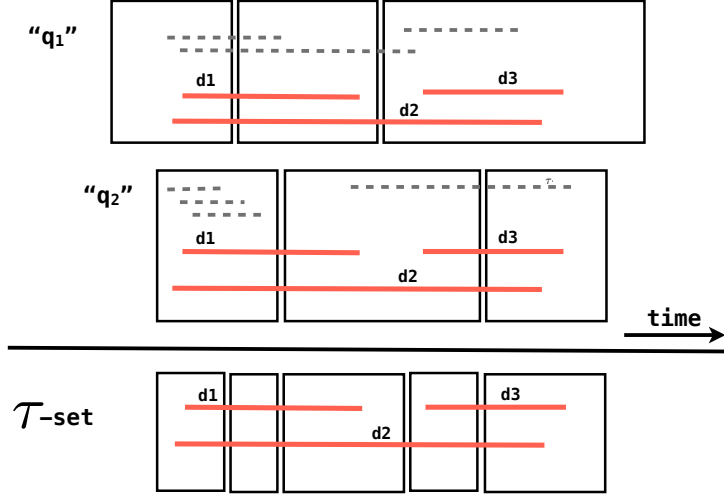


Figure 4.1: τ -set for the affected partitions time-travel query “ $q_1 q_2$ ”

such that each element \mathbf{t} of the resulting tuple is an element of \mathcal{X} has the property that there is non-zero time-overlap between all of the constituent partitions.

$$\mathbf{c} = \{ (c_1, \dots, c_m) \mid \exists c \forall i, b_{c_i} \leq t < e_{c_i} \wedge c_i \in \mathcal{P}_i \}$$

For example, in Figure 4.1, the queries q_1 and q_2 have 3 partitions each with a \mathcal{X} of cardinality 9. However, the resulting τ -set has a cardinality 5 after the τ -join operation.

We further show for Equi-cost based Partition selection, *GreedySelect* chooses elements only from the τ -set. In other words, *GreedySelect* over the cartesian set is equivalent to *GreedySelect* over the constrained τ -set.

Theorem 3. *GreedySelect for Equi-cost based selection on the entire cartesian set \mathcal{X} chooses elements which also belong to the τ -set.*

We prove this theorem by contradiction, by first choosing an element from $\mathcal{X} \setminus \tau$ and showing that we can replace this element with a better candidate from τ . Formally to prove this theorem we need to introduce the notion of *unselected-partition space*. The selection set \bar{S} is the set of already selected partitions and let us for convenience define \bar{S}_d as the actual set of result documents covered by the partitions in \bar{S} . A time range in the intersection-space is said to be *unselected* if none of the partitions in \bar{S} have time-ranges overlapping with the given range. In other words an unselected-space refers

to a range where none of the partitions have been selected, or $\notin \bar{S}$, at the current state of the algorithm.

Benefit of a candidate, introduced earlier, is non-zero if $\bigcap_{x_i \in \mathbf{x}} x_i \setminus \bar{S}_d \neq \emptyset$ and cost of a candidate \mathbf{x} is defined as count of partitions in \mathbf{x} which are unselected. We proceed to prove the following lemma which is essential in the proof for Theorem 3.

Lemma 2. *For any unselected-partition space, the best benefit is given by a candidate from the τ -set.*

Proof. We prove this by contradiction. Let there be a candidate, $\mathbf{x} \in \mathcal{X} \setminus \tau$ which has a higher benefit than any of the candidate $\mathbf{c} \in \tau$. Let $\mathcal{P}_{v,i} \in \mathbf{x}$ has the maximum partition begin time $b_{v,j}$. Using Lemma 1, we can replace partitions of the other terms $\mathcal{P}_{w,j}$ ($w \neq v$) which contain the time $b_{v,j}$ to obtain a higher or equal overlap and hence better benefit. Since the new candidate obtained by the replacement belongs to τ , this is contrary to our assumption hence our claim holds. \square

We now proceed to the proof for the theorem.

Proof. We prove this by induction on the number of iterations i of the greedy-select algorithm.

$i = 1$: For the first iteration the entire intersection-space is unselected, i.e., $\bar{S} = \emptyset$. Given that there is enough budget for selection for m -keyword query i.e. $m \leq IO_BOUND$ we always select partitions from the collapsed set according to lemma 2.

$i \rightarrow i + 1$: Choosing from τ for the first i iterations induces only multiple unselected regions of the intersection-space. Because of the nature of the τ -join certain time-ranges are completely covered/selected and certain ranges are unselected. The partitions which are not in the unselected-space will always have zero benefit hence we can safely discard them.

Now choosing a candidate $\mathbf{x}' \in \mathcal{X} \setminus \tau$ could have a cost value c (where $0 \leq c \leq m$) depending on the number of constituent partitions already in the selection set. To prove that the choice of the candidate is still made from the τ -set we argue as in the proof of Lemma 2. Assume that there is a better candidate $\mathbf{x} \in \mathcal{X} \setminus \tau$ (best benefit/cost ratio), and a non-zero cost c . We can always replace the partitions $x_i \in \mathbf{x} \wedge x_i \notin \bar{S}$ with another partition of the same term in the following ways:

Case 1 – $x_i \notin \bar{S} \quad \forall x_i \in \mathbf{x}$: In the case of \mathbf{x} having no partitions from the selection set \bar{S} , i.e.,

$$\forall x_i \in \mathbf{x} \mid x_i \notin \bar{S}$$

we use lemma 2 to choose a better candidate from τ since there are only unselected-regions from where a choice can be made.

Since we operate only within unselected regions, we denote the minimum time boundary in the region as left region boundary and the maximum time-boundary as the right region boundary. For cases 2 and 3, we consider candidates \mathbf{x} with non-zero benefit, and non-zero cost less than m , i.e.,

$$\exists x_i, \quad x_i \in \mathbf{x} \mid x_i \in \bar{S}.$$

Case 2 – Suppose that the partition $x'_i \in \mathbf{x}' \mid x'_i \in \bar{S}$, only belong to the right region boundary. We can always choose a replacement partition r_j for $x'_j \in \mathbf{x}' \mid x'_j \notin \bar{S}$, where r_j and x'_j belong to the same term, such that the new replacement candidate $\mathbf{r} \in \tau$ has a better benefit than \mathbf{x}' . More specifically, the replacement candidate $\mathbf{r} \in \tau$ has the following selected and unselected partitions

- **selected partitions:** Selected partitions x'_i such that $x'_i \in \mathbf{x}' \mid x'_i \in \bar{S}$.
- **unselected partitions:** Unselected replacement partitions r_j which contain the minimum begin time of the selected partitions, $t_{mbt} = \min\{b_{x'_i} \mid x'_i \in \mathbf{x}' \wedge x'_i \in \bar{S}\}$, i.e.,

$$r_j \mid b_{r_j} \leq t_{mbt} < e_{r_j}$$

It is easy to observe that the replacement candidate \mathbf{r} has the same cost as its counterpart \mathbf{x}' , and a better or equal benefit value, thus giving it an overall better benefit/cost ratio. Since such a replaced candidate belongs to the τ -set, this is contrary to our assumption and our claim holds.

Case 3 – Similar to case 2, if \mathbf{x}' has partitions belonging to the left boundary of the region, we can replace the unselected partitions of each term by a replacement partition which contains/overlaps with the *maximum end time* among the partitions which belong to the selection set in x , i.e., $t_{met} = \max\{e_{x'_i} \mid x'_i \in \mathbf{x}' \wedge x'_i \in \bar{S}\}$. The new replacement candidate \mathbf{r} , set belongs to the τ -set, and has a better or equal benefit than x contrary to our assumption. \square

5 Practical Issues

While the previous two sections presented the theoretical underpinnings for the partition selection problem, in this section, we discuss a few issues during their implementation that we faced in practice and present solutions we used.

Dealing with Partition and Query Boundary Alignment

In our algorithmic descriptions above, we assumed that if a partition overlaps with the query time-range, then its contribution to the final answer set is from *all the entries* in the partition. In other words, we ignored the fact that even within the partition, possibly large number of entries may not satisfy the temporal predicate if the temporal boundaries of the partition are not completely contained within the range specified by the temporal predicate. Note that this affects the estimates of the *benefit* values of the partitions in the boundaries of the query time – thus the benefits of at most 2 partitions per term are in error.

This error can be significantly improved if we adjust the value of benefit of a partition to account for incomplete overlap along the time axis. A straightforward approach for this, which we employ in our implementation, is to *scale* the benefits by the fraction of temporal overlap between the query and the partition. In practice, we observed that this simple scaling (which can be seen to be similar to making uniformity assumption during cardinality estimates) works very well.

I/O Budget Underflow

Another issue that comes up when we are using only *estimates* of benefit provided by partition(s) towards the final answer set is that during partition

selection, we may encounter a situation where *none of the partitions* show any non-zero benefit, although in reality they may contain some results. When faced with such a situation, the partition selection algorithms described in Sections 4 and 3 simply terminate – even if the specified I/O budget allows for more partitions to be read.

To avoid this undesirable behavior, the partition selection algorithm can be modified to ignore the estimates of benefits when *all* the unselected partitions have zero estimated benefits. At this stage, partitions are selected in *decreasing* order of their size as long as the I/O budget is not violated.

6 Evaluation Framework

In this section, we present and discuss the results of a detailed experimental evaluation of our algorithms in terms of their effectiveness in achieve high recall levels while keeping within the specified budget on the index accesses.

6.1 Setup

All our algorithms, including the underlying time-travel inverted index framework, were implemented using Java 1.6. All experiments were conducted on Dell PowerEdge M610 servers with 2 Intel Xeon E5530 CPUs, 48 GB of main memory, a large iSCSI-attached disk array, and Debian GNU/Linux (SMP Kernel 2.6.29.3.1) as operating system. Experiments were conducted using the Java Hotspot 64-Bit Server VM (build 11.2-b01).

6.2 Datasets and Query Workload

For our experiments we used three different datasets, all derived from real-world data sources.

WIKI The *English Wikipedia revision history* [12], whose uncompressed raw data amounts to 0.7 TBytes, contains the full editing history of the English Wikipedia from January 2001 to December 2005. We indexed all versions of encyclopedia articles excluding versions that were marked as the result of a minor edit (e.g., the correction of spelling errors etc.). This yielded a total of 1,517,524 documents with 15,079,829 versions having a mean (μ) of 9.94 versions per document at standard deviation (σ) of 46.08.

UKGOV This is a subset of the European Archive [1], containing weekly crawls of the eleven governmental websites from the U.K. We filtered

out documents not belonging to MIME-types `text/plain` and `text/html` to obtain a dataset that totals 0.4 TBytes. This dataset includes 685,678 documents with 17,297,548 versions ($\mu = 25.23$ and $\sigma = 28.38$).

NYT The *New York Times Annotated corpus* [2] comprises more than 1.8 million articles from the New York Times published between 1987 and 2007. Every article has an associated time-stamp which was taken as the begin time for that article. The end time for each article was chosen to be 90 days after the begin time, giving every document a validity time of 90 days. This is done to reflect the real world setting where the news articles are publicly available only for a limited period from their publication, and also to coarsely model the commonly used time-decaying relevance model for news articles.

Note that each of these datasets represents a realistic class of time varying text collection typically used in temporal text analytics. Specifically, WIKI corresponds to an explicitly version controlled text collection, UKGOV is an archive of the evolving Web, and NYT is an instance of archive of continually generated newspaper content. For the ease of experimentation, we rounded the time-stamps of versions to the nearest day for all datasets.

We compiled three dataset-specific query workloads by extracting frequent queries from the AOL query logs, which were temporarily made available during 2006. For the WIKI dataset we extracted 300 most frequent queries which had a result click on the domain `en.wikipedia.org` and similarly for NYT and UKGOV we compiled 300 queries which had a result hit on `nytimes.com` and 50 queries which had result hit on `.gov.uk` domains. Using these keyword queries, we generated a time-travel query workload with 3 instances each for the following 2 different temporal predicate granularities: 30 days and 1 year.

6.3 Index Management

The time-travel inverted index is stored on disk using flat files containing both the lexicon as well as inverted lists. At run time, the lexicon is read completely into memory, and for a given query the appropriate partition is retrieved from the index flat file on disk. These inverted lists are stored using 7-Bit compression. The synopses of partitions were maintained in a separate flat file in a similar fashion.

For temporal partitioning of the index, we employ a very simple approach in which a partition boundary is placed after a fixed time window. We avoided using more sophisticated partitioning strategies from [7] as they

Index	UKGOV	NYT	WIKI
Fixed-7	11G	13G	13G
Synopsis Index - 5% sample	146MB	134MB	146MB
Synopsis Index - 10% sample	291MB	258MB	290MB
Fixed-30	4.4G	3.5G	6.3G
Synopsis Index - 5% sample	61MB	39MB	75MB
Synopsis Index - 10% sample	122MB	74MB	149MB

Table 6.1: Synopsis Index

can not be easily maintained incrementally, and also due to their high computational overheads. We present results for two time window sizes: (i) 1 week (referred to as **Fixed-7** partitioning), and (ii) 1 month (referred to as **Fixed-30** partitioning). Unless otherwise mentioned, all the results presented in this paper are from **Fixed-7** partitioning. We also present results from index structures built using [7]). More specifically, we build index structures using the space-bound approach with the parameters $\kappa = 1.5, 2.5$ as two representatives of lower and higher degree of partitioning. These are represented as **SB 1.5** and **SB 2.5** respectively.

The estimates from the KMV synopses [8] that we chose to implement are naturally dependent on their size in relation to the base data size. We experimented with two sizes of synopses: 5% and 10% of the partition size (with minimum size set to 100). Unless otherwise mentioned, we report results for 10% size of the KMV synopsis. A synopsis index was generated during index construction time and stored as flat files on disk. Instead of storing the list of hashed double values of the KMV synopsis, the corresponding document identifiers(integers) were stored for better compression (Table 6.1). The doc ids were translated to their respective doubles during query time for the necessary KMV intersection estimation.

Finally, we employed a practically infeasible **oracle** for partition selection, which computes the *ideal values* of set operations (intersection and union) between partitions. Oracle computes these values by simply evaluating the query completely, without any partition selection, and then uses them in partition selection to overcome the errors due to estimates from the KMV synopses.

6.4 Evaluation Methodology

We conducted experiments aimed at evaluating the effectiveness of partition selection, in terms of the recall obtained for the final answer set for each query, as we vary the specified I/O budget. The budget bounds were incremented in steps of 0.1, starting from 0, and the recall values obtained for each instance of the time-travel keyword query were averaged. During averaging, we ignored time-travel keyword queries that affect only one partition each of the terms involved. We also ignored queries which have no results as they contribute to false-positives for partition selection.

For comparing I/O performance of different techniques, we measure the number of index entries read after applying the partition selection – denoted as **RWS**, and the number of index entries read without applying partition selection – denoted as **RWOS**. The ratio $\frac{RWS}{RWOS}$, called *Ratio of Index Reads*, is denoted as **RIR**. We also measure the actual query runtimes to show their correspondence with the *Ratio of Index Reads*.

7 Experimental Results

7.1 Performance of Partition Selection

In the first set of experiments, we demonstrate the impact of using partition selection in identifying the set of partitions that maximize the final result recall, while adhering to the specified I/O budget. As described before, the I/O budget can be specified in two forms – based on size of partitions, or based on the number of partitions. For both the budget formulations, we ran the full query workload for each of the datasets, and present aggregated results individually for every query time-window. The results are plotted in Figure 7.1 for size-based selection, and in Figure 7.2 for equi-cost partition selection approaches.

These graphs unequivocally demonstrate that the partition selection can be very effective in speeding up the time-travel query processing with minimal impact on the quality of final results.

Going further, we notice that for queries with time-window of a month, the selection algorithm selects the most relevant partition thus providing high levels of recall by reading close to 50% of the affected entries/partitions. Similar behavior is also seen for Equi-cost partition selection, which manages to read the correct set of partitions to obtain as high a recall as possible.

In case of queries with yearly time-window, relevant entries are spread over a larger number of partitions. This allows for a greater flexibility in choosing the partitions for processing. This allows selection algorithms in both Equi-cost and Size-based variations, to report higher quality results even with very low I/O budget.

Overall, from these results, one can observe that the partition selection under either size-based or equi-cost model show very similar performance behavior. Therefore, we omit the results of equi-cost partition selection in the rest of the section.

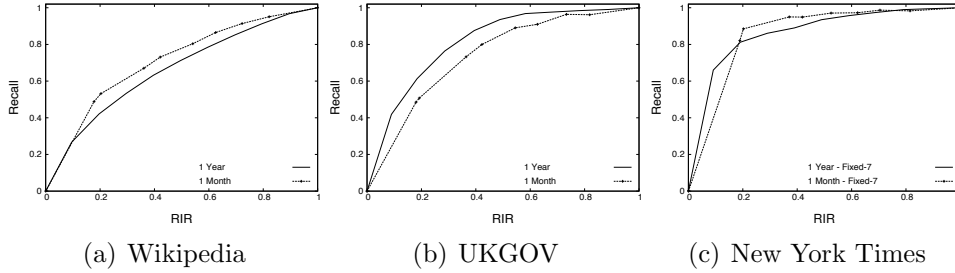


Figure 7.1: Performance of Size-based Partition Selection with Different Query Time-windows on Fixed-7 Index

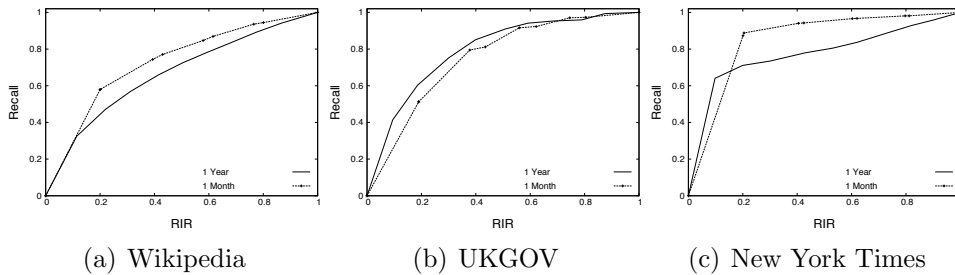


Figure 7.2: Performance of Equi-cost Partition Selection with Different Query Time-windows on Fixed-7 Index

7.2 Query Runtimes

The focus was on measuring runtimes in a cold-cache setup. We start with a cold-cache and flush it after each query execution step. Each time-travel query from the workload was evaluated for 10 I/O bounds (0.1 through to 1.0) and the average time taken (in milliseconds) for each of these bounds are presented in Tables 7.1, 7.2 and 7.3. The first column represents the tunable input I/O bound parameter β , which indicates the fraction of the affected entries to be read. This is followed by the reporting the recall attained along with the average runtimes for the specified bound. We compare the results of selection based retrieval with two competitors: (a) the standard unpartitioned inverted index list, **unpartitioned**, and (b) partitioned lists not supporting partition selection, **no – sel**.

The reported runtime for each query is the sum of time taken by the synopsis based query plan computation (partition selection) and the actual query processing time with the selected partitions. We exclude the query plan computation time for the specific bound 1.0 in the results since the bound eventually results in selection of all the partitions. The runtime for this bound incidentally is also the time taken for **no – sel**. The runtime

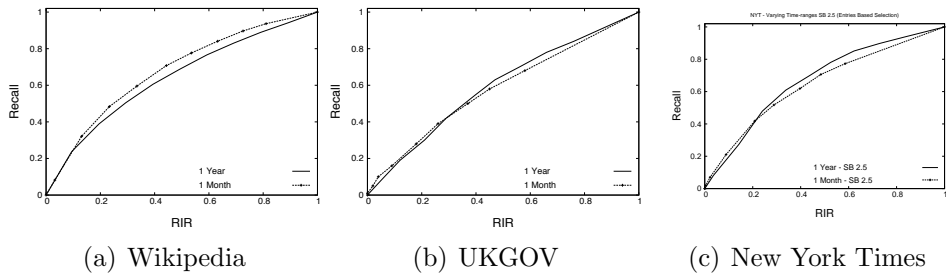


Figure 7.3: Performance of Size-based Partition Selection with Different Query Time-windows on SB ($\kappa = 2.5$) Index

results further corroborate the observations presented before. Observe that in case of executing time-travel queries against an unpartitioned index takes almost 3 secs for WIKI in Table 7.1, 1 sec for NYT in Table 7.3 and as large as 12 secs for UKGOV in Table 7.2 irrespective of the query time-range. Having a partitioned index improves on this as is indicated by the **no – sel** values in the tables. However partition selection over such an partitioned index further reduces execution time to give recall values of almost 0.8 in only 50%-60% time. In case of comparatively smaller collections, say NYT, one might potentially argue about the performance not being significant in terms of absolute runtimes but in larger corpora (like UKGOV) it can make a significant difference in performance as is shown in Table 7.2.

The anytime nature of the selection algorithm also means that the user can terminate the query processing at any instant she wishes and can still get the maximum recall at that stage of the computation. A quick preview at the results after $3/4$ of a second can prove beneficial with almost 90% recall (UKGOV monthly query) or 85% recall (WIKI yearly query). The results for space bound indexing follow a similar trend (as reported in Figures 7.4 to 7.6)

7.3 Impact of using Synopses

The first set of experiments were aimed at quantifying the impact of using KMV synopses for the estimation of benefits, and the effect of different synopses size. For each dataset, we measure the average recall obtained for each granularity of time-travel queries, using 5% and 10% synopses, and compare them with those of idealized *oracle* outlined earlier. The results of this experiments over indexes with Fixed-7 partitioning, are shown in Figure 7.4, for queries with temporal predicates of 1 year and 1 week ranges.

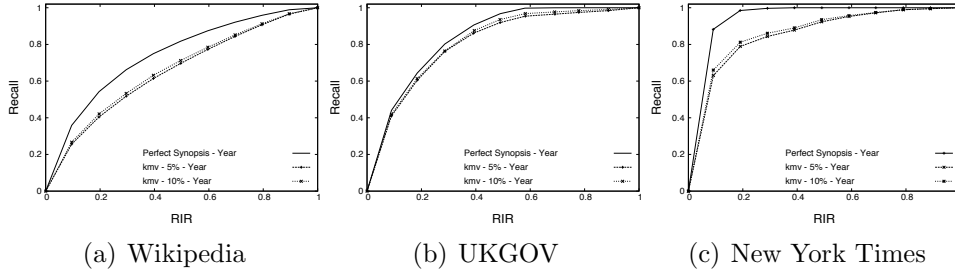


Figure 7.4: Impact of Using Synopses

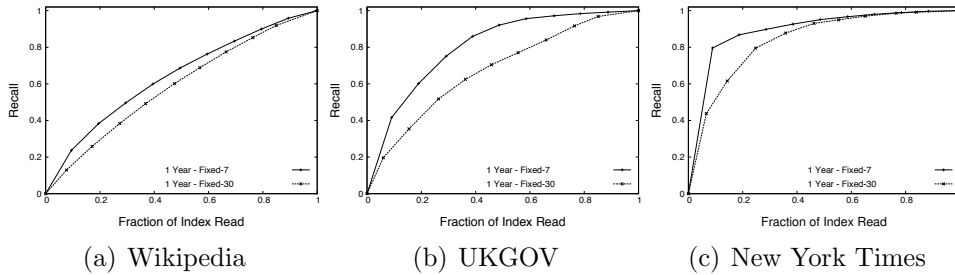


Figure 7.5: Effect of varying Partitioning Granularities for Queries with 1-year Time-window

We can make the following observations from these plots: (i) The gap between a 5% KMV synopsis and 10% synopsis is negligibly small, prompting our choice of using 5% KMV synopsis. (ii) Although oracle based estimates are, as expected, better overall, improvements over using KMV synopsis estimates are not significant. KMV synopsis are stored as arrays of doubles and much smaller than individual postings. Moreover the estimate computation is fast and they can be compressed and kept in memory for computing the selection set efficiently.

These results also provide a first glimpse of the effectiveness of the partition selection methods themselves – in the best case for NYT dataset, partition selection methods are able to answer with more than 80% recall when the I/O budget is as small as 20% of the RWOS.

7.4 Impact of Partition Granularity

We experimented with two different granularities of partitioning – viz., 7-day and 30-day time-ranges, resulting in Fixed-7 and Fixed-30 index configurations. Fixed-7 has a higher number of partitions, thus can be seen as having smaller partition sizes in comparison to Fixed-30. Clearly, this allows efficient

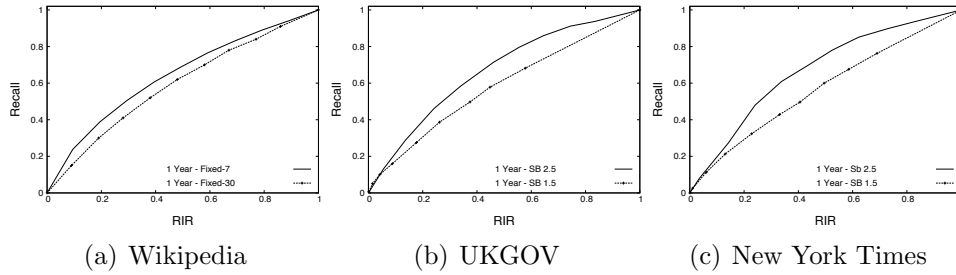


Figure 7.6: Effect of varying Partitioning Granularities for Queries with 1-year Time-window on SB ($\kappa = 2.5$)Index

processing of time-point or short duration queries. However, it deteriorates for larger time-range queries if no partition selection is made. On the other hand, performance with partition selection shown in Figure 7.5 for queries with 1-year time-window, shows that even for smaller partitions sizes this issue can be effectively alleviated. Similar results are seen for space bound partitioning approaches (Figure 7.6).

WIKI- Unpartitioned time: 3,217 ms				
Bound	Yearly no-sel : 1,020.77 ms		Monthly no-sel: 212.01 ms	
	Recall	Av. Time (ms)	Recall	Av. Time (ms)
0.1	0.27	207.6	0.01	5.7
0.2	0.42	367.7	0.49	88.5
0.3	0.53	436.8	0.53	94.3
0.4	0.63	521.0	0.67	134.3
0.5	0.71	594.8	0.73	135.9
0.6	0.78	646.7	0.80	165.2
0.7	0.85	736.3	0.87	165.9
0.8	0.91	798.5	0.91	186.4
0.9	0.97	877.4	0.95	192.0
1	1.00	1,020.8	1.00	212.0

Table 7.1: Query Runtimes - Wikipedia

UKGOV- Unpartitioned time: 12,598ms				
Bound	Yearly no-sel : 8,490 ms		Monthly no-sel: 1,225 ms	
	Recall	Av. Time (ms)	Recall	Av. Time (ms)
0.1	0.42	1,615.9	0.00	0
0.2	0.61	2,788.8	0.48	352.6
0.3	0.76	3,705.2	0.51	370.4
0.4	0.88	4,592.1	0.73	590.3
0.5	0.94	5,183.2	0.80	644.3
0.6	0.97	5,772.6	0.89	751.2
0.7	0.98	6,427.9	0.91	852.7
0.8	0.98	7,025.2	0.96	927.4
0.9	0.99	7,635.8	0.96	1,026.6
1	1.00	8,490.1	1.00	1,225.9

Table 7.2: Query Runtimes - UKGOV

NYT - Unpartitioned time: 1,014 ms				
Bound	Yearly no-sel : 525.7346 ms		Monthly no-sel: 146 ms	
	Rec	Av. Time (ms)	Rec	Av. Time (ms)
0.1	0.66	200.9	0.00	0
0.2	0.81	254.5	0.82	103.5
0.3	0.86	301.4	0.89	106
0.4	0.89	308.2	0.95	117.5
0.5	0.94	328.1	0.95	122
0.6	0.96	358.7	0.97	125
0.7	0.97	384.8	0.97	126
0.8	0.99	428.6	0.99	128.5
0.9	0.99	468.2	0.98	141.5
1	1.00	525.7	1.00	146

Table 7.3: Query Runtimes - NYT

UKGOV - Unpartitioned time: 12,598				
Bound	Yearly		Monthly	
	no-sel : 2,720 ms		no-sel: 572 ms	
	Recall	Avg. Time (ms)	Recall	Avg. Time (ms)
0.1	0.13	295.1	0.00	2.3
0.2	0.29	601.6	0.08	41.8
0.3	0.46	915.6	0.26	124.9
0.4	0.58	1260.7	0.38	172.3
0.5	0.72	1544.4	0.54	239.7
0.6	0.80	1803.1	0.59	271.1
0.7	0.86	2098.3	0.65	325.0
0.8	0.91	2265.2	0.72	358.7
0.9	0.94	2436.2	0.80	395.7
1	1.00	2720.2	1.00	572.7

Table 7.4: Query Runtimes - UKGOV - SB ($\kappa = 2.5$)

WIKI - Unpartitioned time: 3,217				
Bound	Yearly		Monthly	
	no-sel : 596.46 ms		no-sel: 129.06 ms	
	Recall	Avg. Time (ms)	Recall	Avg. Time (ms)
0.1	0.23	105.8	0.08	12.8
0.2	0.38	154.4	0.32	33.4
0.3	0.50	212.2	0.48	44.0
0.4	0.60	268.1	0.59	55.5
0.5	0.68	320.5	0.70	64.6
0.6	0.76	375.6	0.77	75.6
0.7	0.83	429.4	0.84	87.4
0.8	0.88	477.6	0.89	97.2
0.9	0.94	539.0	0.93	104.3
1	1	596.4	1	129.0

Table 7.5: Query Runtimes - WIKI - SB ($\kappa = 2.5$)

NYT - Unpartitioned time: 1,014 ms				
Bound	Yearly		Monthly	
	no-sel : 83.17 ms		no-sel: 82.04 ms	
	Recall	Avg. Time (ms)	Recall	Avg. Time (ms)
0.1	0.08	26.7	0	0
0.2	0.27	53.0	0.01	1
0.3	0.47	50.87	0.07	3
0.4	0.60	54.9	0.21	11.8
0.5	0.69	60.6	0.41	21.4
0.6	0.78	65.3	0.51	19.9
0.7	0.85	72.4	0.61	24.1
0.8	0.89	81.9	0.70	26.1
0.9	0.93	83.6	0.77	27.2
1	1	90.1	1	44.7

Table 7.6: Query Runtimes - NYT - SB ($\kappa = 2.5$)

8 Related Work

Research in Information Retrieval has recently paid attention to temporal information associated with documents. Alonso et al. [5] give an overview of relevant research directions. Closest to the ideas presented here is the work on time-travel text search [7] that allows users to search only the part of a document collection that existed at a given time point. To support this functionality efficiently, posting lists from an inverted index are temporally partitioned either according to a given space bound or required performance guarantee. Postings whose valid-time interval overlaps with multiple of the determined temporal partitions are judiciously replicated and put into multiple posting lists, thus increasing the overall size of the index.

Whereas the related research discussed thus far focuses on textual documents as one specific type of data, research in temporal databases has taken a broader perspective and targeted general data that comes with attached temporal information. Index structures tailored to such data like the Multi-Version B-Tree [6] or LHAM [11] are related to the present work, since they also, implicitly or explicitly, rely on a temporal partitioning and replication of data. It is therefore conceivable to apply our proposed techniques in conjunction with one of these index structures. Join processing techniques for temporal databases [9] are a second class of related work whose focus, to the best of our knowledge, has been on producing accurate query results opposed to the approximate results that our techniques deliver.

As data volumes grow, many queries are increasingly expensive to evaluate accurately. However, an approximate but almost accurate answer that is delivered quickly is often good enough. Approximate query processing techniques [3, 4] developed by the database community aim at quickly determining an approximate answer and, to this end, typically leverage data statistics (often approximated using histograms), sampling, and other data synopses. In contrast to our scenario, approximate query processing techniques target scenarios with a well-designed relational schema that implies certain reasonable queries (e.g., based on foreign keys). When cast into a re-

lational schema, our scenario gives rise to millions of relations (corresponding to terms and their corresponding partitions).

9 Conclusions and Future Work

This work presented techniques to efficiently process time-travel queries with temporally partitioned inverted indexes. By carefully selecting partitions of the index which contribute most to recall at any stage of the processing, our methods reduce the number of duplicate reads of the same items. Our experimental results showed that recall levels of at least 80% can be achieved by reading only 40% of the index entries, significantly reducing query processing time. This is particularly useful in text analytics with multiple rounds of query reformulations where fast retrieval of a representative subset of results is needed. This work opens up interesting questions for future research, e.g.: How to organize index structures so that only essential entries are read, thus improving efficiency? How to apply partition selection techniques to index structures which have overlapping partitions? How to further improve query processing by encoding and skipping techniques?

Bibliography

- [1] European archive. <http://www.europarchive.org>.
- [2] New york times annotated corpus. <http://corpus.nytimes.com>.
- [3] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A Fast Decision Support Systems Using Approximate Query Answers. In *VLDB*, pages 754–757, 1999.
- [4] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, pages 275–286, 1999.
- [5] O. Alonso, M. Gertz, and R. Baeza-Yates. On the value of temporal information in information retrieval. *SIGIR Forum*, 41(2):35–41, 2007.
- [6] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. An asymptotically optimal multiversion b-tree. *The VLDB Journal*, 5(4), 1996.
- [7] K. Berberich, S. Bedathur, T. Neumann, and G. Weikum. A Time Machine for Text Search. In *SIGIR*, 2007.
- [8] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *SIGMOD*, pages 199–210, 2007.
- [9] D. Gao, C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Join operations in temporal databases. *The VLDB Journal*, 14(1):2–29, 2005.
- [10] S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.

- [11] P. Muth, P. E. O’Neil, A. Pick, and G. Weikum. The LHAM Log-Structured History Data Access Method. *VLDB J.*, 8(3-4):199–221, 2000.
- [12] wikipedia. <http://en.wikipedia.org/>.