A Time Machine for Text Search

Klaus Berberich Srikanta Bedathur Thomas Neumann Gerhard Weikum

Max-Planck Institute for Informatics
Saarbrücken, Germany
{kberberi, bedathur, neumann, weikum}@mpi-inf.mpg.de

ABSTRACT

Text search over temporally versioned document collections such as web archives has received little attention as a research problem. As a consequence, there is no scalable and principled solution to search such a collection as of a specified time t. In this work, we address this shortcoming and propose an efficient solution for *time-travel text search* by extending the inverted file index to make it ready for temporal search. We introduce approximate temporal coalescing as a tunable method to reduce the index size without significantly affecting the quality of results. In order to further improve the performance of time-travel queries, we introduce two principled techniques to trade off index size for its performance. These techniques can be formulated as optimization problems that can be solved to near-optimality. Finally, our approach is evaluated in a comprehensive series of experiments on two large-scale real-world datasets. Results unequivocally show that our methods make it possible to build an efficient "time machine" scalable to large versioned text collections.

Categories and Subject Descriptors

H.3.1 [**Content Analysis and Indexing**]: Indexing methods; H.3.3 [**Information Search and Retrieval**]: Retrieval models, Search process

General Terms

Algorithms, Experimentation, Performance

Keywords

Temporal text indexing, time-travel text search, web archives

1. INTRODUCTION

In this work we address *time-travel text search* over temporally versioned document collections. Given a keyword query q and a time t our goal is to *identify and rank relevant documents as if the collection was in its state as of time t.*

An increasing number of such versioned document collections is available today including web archives, collaborative authoring environments like Wikis, or timestamped information feeds. Text search on these collections, however, is mostly time-ignorant: while the searched collection changes over time, often only the most recent version of a documents is indexed, or, versions are indexed independently and treated as separate documents. Even worse, for some collections, in particular web archives like the Internet Archive [18], a comprehensive text-search functionality is often completely missing.

Time-travel text search, as we develop it in this paper, is a crucial tool to explore these collections and to unfold their full potential as the following example demonstrates. *For a documentary about a past political scandal, a journalist needs to research early opinions and statements made by the involved politicians. Sending an appropriate query to a major web search-engine, the majority of returned results contains only recent coverage, since many of the early web pages have disappeared and are only preserved in web archives. If the query could be enriched with a time point, say August 20th 2003 as the day after the scandal got revealed, and be issued against a web archive, only pages that existed specifically at that time could be retrieved thus better satisfying the journalist's information need.*

Document collections like the Web or Wikipedia [32], as we target them here, are already large if only a single snapshot is considered. Looking at their evolutionary history, we are faced with even larger data volumes. As a consequence, naïve approaches to time-travel text search fail, and viable approaches must scale-up well to such large data volumes.

This paper presents an efficient solution to time-travel text search by making the following key contributions:

1. The popular well-studied *inverted file index* [35] is transparently extended to enable time-travel text search.
2. *Temporal coalescing* is introduced to avoid an index-size explosion while keeping results highly accurate.
3. We develop two *sublist materialization* techniques to improve index performance that allow trading off space vs. performance.
4. In a *comprehensive experimental evaluation* our approach is evaluated on the English Wikipedia and parts of the Internet Archive as two large-scale real-world datasets with versioned documents.

The remainder of this paper is organized as follows. The presented work is put in context with related work in Section 2. We delineate our model of a temporally versioned document collection in Section 3. We present our time-travel inverted index in Section 4. Building on it, temporal coalescing is described in Section 5. In Section 6 we describe principled techniques to improve index performance, before presenting the results of our experimental evaluation in Section 7.

SIGIR'07, July 23–27, 2007, Amsterdam, The Netherlands.

2. RELATED WORK

We can classify the related work mainly into the following two categories: (i) methods that deal explicitly with collections of versioned documents or temporal databases, and (ii) methods for reducing the index size by exploiting either the document-content overlap or by pruning portions of the index. We briefly review work under these categories here.

To the best of our knowledge, there is very little prior work dealing with historical search over temporally versioned documents. Anick and Flynn [3], while pioneering this research, describe a help-desk system that supports historical queries. Access costs are optimized for accesses to the most recent versions and increase as one moves farther into the past. Burrows and Hisgen [10], in a patent description, delineate a method for indexing range-based values and mention its potential use for searching based on dates associated with documents. Recent work by Nørvåg and Nybø [25] and their earlier proposals concentrate on the relatively simpler problem of supporting text-containment queries only and neglect the relevance scoring of results. Stack [29] reports practical experiences made when adapting the open source search-engine Nutch to search web archives. This adaptation, however, does not provide the intended time-travel text search functionality. In contrast, research in temporal databases has produced several index structures tailored for time-evolving databases; a comprehensive overview of the state-of-art is available in [28]. Unlike the inverted file index, their applicability to text search is not well understood.

Moving on to the second category of related work, Broder et al. [8] describe a technique that exploits large content overlaps between documents to achieve a reduction in index size. Their technique makes strong assumptions about the structure of document overlaps rendering it inapplicable to our context. More recent approaches by Hersovici et al. [17] and Zhang and Suel [34] exploit arbitrary content overlaps between documents to reduce index size. None of the approaches, however, considers time explicitly or provides the desired time-travel text search functionality. Static index-pruning techniques [11, 12] aim to reduce the effective index size, by removing portions of the index that are expected to have low impact on the query result. They also do not consider temporal aspects of documents, and thus are technically quite different from our proposal despite having a shared goal of index-size reduction. It should be noted that index-pruning techniques can be adapted to work along with the temporal text index we propose here.

3. MODEL

In the present work, we deal with a temporally versioned document collection $\mathbf{D}$ that is modeled as described in the following. Each document $\mathbf{d} \in \mathbf{D}$ is a sequence of its versions

$$\mathbf{d} = \langle d^{\,t_1}, d^{\,t_2}, \ldots \rangle \, .$$

Each version $d^{\,t_i}$ has an associated timestamp t_i reflecting when the version was created. Each version is a vector of searchable terms or features. Any modification to a document version results in the insertion of a new version with corresponding timestamp. We employ a discrete definition of time, so that timestamps are non-negative integers. The deletion of a document at time t_i, i.e., its disappearance from the current state of the collection, is modeled as the insertion of a special “tombstone” version $\perp$. The validity time-interval $val(d^{\,t_i})$ of a version $d^{\,t_i}$ is $[t_i, t_{i+1})$, if a newer version with associated timestamp t_{i+1} exists, and $[t_i, now)$ otherwise where *now* points to the greatest possible value of a timestamp (i.e., $\forall t \, : \, t < now$).

Putting all this together, we define the state $\mathbf{D}^{\,t}$ of the collection at time t (i.e., the set of versions valid at t that are not deletions) as

$$\mathbf{D}^{\,t} = \bigcup_{\mathbf{d} \in \mathbf{D}} \{ d^{\,t_i} \in \mathbf{d} \mid t \in val(d^{\,t_i}) \, \wedge \, d^{\,t_i} \neq \perp \} \, .$$

As mentioned earlier, we want to enrich a keyword query q with a timestamp t, so that q be evaluated over $\mathbf{D}^{\,t}$, i.e., the state of the collection at time t. The enriched *time-travel query* is written as $q^{\,t}$ for brevity.

As a retrieval model in this work we adopt Okapi BM25 [27], but note that the proposed techniques are not dependent on this choice and are applicable to other retrieval models like tf-idf [4] or language models [26] as well. For our considered setting, we slightly adapt Okapi BM25 as

$$w(q^{\,t}, d^{\,t_i}) = \sum_{v \in q} w_{tf}(v, d^{\,t_i}) \cdot w_{idf}(v, t) \, .$$

In the above formula, the relevance $w(q^{\,t}, d^{\,t_i})$ of a document version $d^{\,t_i}$ to the time-travel query $q^{\,t}$ is defined. We reiterate that $q^{\,t}$ is evaluated over $\mathbf{D}^{\,t}$ so that only the version $d^{\,t_i}$ valid at time t is considered. The first factor $w_{tf}(v, d^{\,t_i})$ in the summation, further referred to as the *tf-score* is defined as

$$w_{tf}(v, d^{\,t_i}) = \frac{(k_1 + 1) \cdot tf(v, d^{\,t_i})}{k_1 \cdot ((1 - b) + b \cdot \frac{dl(d^{\,t_i})}{avdl(t_i)}) + tf(v, d^{\,t_i})} \, .$$

It considers the plain term frequency $tf(v, d^{\,t_i})$ of term v in version $d^{\,t_i}$ normalizing it, taking into account both the length $dl(d^{\,t_i})$ of the version and the average document length $avdl(t_i)$ in the collection at time t_i. The length-normalization parameter b and the tf-saturation parameter k_1 are inherited from the original Okapi BM25 and are commonly set to values 1.2 and 0.75 respectively. The second factor $w_{idf}(v, t)$, which we refer to as the *idf-score* in the remainder, conveys the inverse document frequency of term v in the collection at time t and is defined as

$$w_{idf}(v, t) = log \, \frac{N(t) - df(v, t) + 0.5}{df(v, t) + 0.5}$$

where $N(t) = |\mathbf{D}^{\,t}|$ is the collection size at time t and $df(v, t)$ gives the number of documents in the collection that contain the term v at time t. While the idf-score depends on the whole corpus as of the query time t, the tf-score is specific to each version.

4. TIME-TRAVEL INVERTED FILE INDEX

The inverted file index is a standard technique for text indexing, deployed in many systems. In this section, we briefly review this technique and present our extensions to the inverted file index that make it ready for time-travel text search.

4.1 Inverted File Index

An inverted file index consists of a *vocabulary*, commonly organized as a B$^+$-Tree, that maps each term to its *idf-score* and *inverted list*. The index list L_v belonging to term v contains *postings* of the form

$$(\, d, \, p \,)$$

where d is a document-identifier and p is the so-called payload. The payload p contains information about the term frequency of v in d, but may also include positional information about where the term appears in the document.

The sort-order of index lists depends on which queries are to be supported efficiently. For Boolean queries it is favorable to sort index lists in document-order. Frequency-order and impact-order sorted index lists are beneficial for ranked queries and enable optimized query processing that stops early after having identified the k most relevant documents [1, 2, 9, 15, 31]. A variety of compression techniques, such as encoding document identifiers more compactly, have been proposed [33, 35] to reduce the size of index lists. For an excellent recent survey about inverted file indexes we refer to [35].

4.2 Time-Travel Inverted File Index

In order to prepare an inverted file index for time travel we extend both inverted lists and the vocabulary structure by explicitly incorporating temporal information. The main idea for inverted lists is that we include a validity time-interval $[t_b, t_e)$ in postings to denote when the payload information was valid. The postings in our time-travel inverted file index are thus of the form

$$(\, d,\, p,\, [t_b, t_e)\,)$$

where d and p are defined as in the standard inverted file index above and $[t_b, t_e)$ is the validity time-interval.

As a concrete example, in our implementation, for a version $d^{\,t_i}$ having the Okapi BM25 tf-score $w_{tf}(v, d^{\,t_i})$ for term v, the index list L_v contains the posting

$$(\, d,\, w_{tf}(v, d^{\,t_i}),\, [t_i, t_{i+1})\,)\ .$$

Similarly, the extended vocabulary structure maintains for each term a time-series of idf-scores organized as a B$^+$-Tree. Unlike the tf-score, the idf-score of every term could vary with every change in the corpus. Therefore, we take a simplified approach to idf-score maintenance, by computing idf-scores for all terms in the corpus at specific (possibly periodic) times.

4.3 Query Processing

During processing of a time-travel query $q^{\,t}$, for each query term the corresponding idf-score valid at time t is retrieved from the extended vocabulary. Then, index lists are sequentially read from disk, thereby accumulating the information contained in the postings. We transparently extend the sequential reading, which is – to the best of our knowledge – common to all query processing techniques on inverted file indexes, thus making them suitable for time-travel query-processing. To this end, sequential reading is extended by skipping all postings whose validity time-interval does not contain t (i.e., $t \notin [t_b, t_e)$). Whether a posting can be skipped can only be decided after the posting has been transferred from disk into memory and therefore still incurs significant I/O cost. As a remedy, we propose index organization techniques in Section 6 that aim to reduce the I/O overhead significantly.

We note that our proposed extension of the inverted file index makes no assumptions about the sort-order of index lists. As a consequence, existing query-processing techniques and most optimizations (e.g., compression techniques) remain equally applicable.

5. TEMPORAL COALESCING

If we employ the time-travel inverted index, as described in the previous section, to a versioned document collection, we obtain one posting per term per document version. For frequent terms and large highly-dynamic collections, this leads to extremely long index lists with very poor query-processing performance.

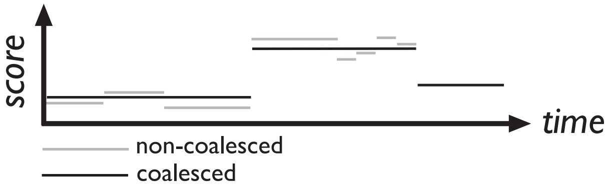


Figure 1: Approximate Temporal Coalescing

The *approximate temporal coalescing* technique that we propose in this section counters this blowup in index-list size. It builds on the observation that most changes in a versioned document collection are minor, leaving large parts of the document untouched. As a consequence, the payload of many postings belonging to temporally adjacent versions will differ only slightly or not at all. Approximate temporal coalescing reduces the number of postings in an index list by merging such a sequence of postings that have almost equal payloads, while keeping the maximal error bounded. This idea is illustrated in Figure 1, which plots non-coalesced and coalesced scores of postings belonging to a single document. Approximate temporal coalescing is greatly effective given such fluctuating payloads and reduces the number of postings from 9 to 3 in the example. The notion of temporal coalescing was originally introduced in temporal database research by Böhlen et al. [6], where the simpler problem of coalescing only equal information was considered.

We next formally state the problem dealt with in approximate temporal coalescing, and discuss the computation of optimal and approximate solutions. Note that the technique is applied to each index list separately, so that the following explanations assume a fixed term v and index list L_v.

As an input we are given a sequence of temporally adjacent postings

$$I = \langle\ (\, d,\, p_i,\, [t_i, t_{i+1})\,)\,), \ldots, (\, d,\, p_{n-1},\, [t_{n-1}, t_n))\,)\ \rangle\ .$$

Each sequence represents a contiguous time period during which the term was present in a single document d. If a term disappears from d but reappears later, we obtain multiple input sequences that are dealt with separately. We seek to generate the *minimal length* output sequence of postings

$$O = \langle\ (\, d,\, p_j,\, [t_j, t_{j+1})\,)\,), \ldots, (\, d,\, p_{m-1},\, [t_{m-1}, t_m))\,)\ \rangle\ ,$$

that adheres to the following constraints: First, O and I must cover the same time-range, i.e., $t_i = t_j$ and $t_n = t_m$. Second, when coalescing a subsequence of postings of the input into a single posting of the output, we want the approximation error to be below a threshold ϵ. In other words, if $(d,\, p_i,\, [t_i, t_{i+1}))$ and $(d,\, p_j,\, [t_j, t_{j+1}))$ are postings of I and O respectively, then the following must hold for a chosen *error* function and a threshold ϵ:

$$t_j \le t_i\ \wedge\ t_{i+1} \le t_{j+1} \Rightarrow error(p_i, p_j) \le \epsilon\ .$$

In this paper, as an *error* function we employ the relative error between payloads (i.e., tf-scores) of a document in I and O, defined as:

$$err_{rel}(p_i, p_j) = |p_i - p_j| \,/\, |p_i|\ .$$

Finding an optimal output sequence of postings can be cast into finding a piecewise-constant representation for the points (t_i, p_i) that uses a minimal number of segments while retaining the above approximation guarantee. Similar problems occur in time-series segmentation [21, 30] and histogram construction [19, 20]. Typically dynamic programming is applied to obtain an optimal solution in $O(n^2\, m^*)$ [20, 30] time with m^* being the number of segments in an optimal sequence. In our setting, as a key difference, only a guarantee on the local error is retained – in contrast to a guarantee on the global error in the aforementioned settings. Exploiting this fact, an optimal solution is computable by means of *induction* [24] in $O(n^2)$ time. Details of the optimal algorithm are omitted here but can be found in the accompanying technical report [5].

The quadratic complexity of the optimal algorithm makes it inappropriate for the large datasets encountered in this work. As an alternative, we introduce a linear-time approximate algorithm that is based on the sliding-window algorithm given in [21]. This algorithm produces nearly-optimal output sequences that retain the bound on the relative error, but possibly require a few additional segments more than an optimal solution.

Algorithm 1 Temporal Coalescing (Approximate)

```
$I = \langle\ (d,\, p_i,\, [t_i, t_{i+1})\,), \ldots \rangle \quad O = \langle\,\rangle$
$p_{min} = p_i \quad p_{max} = p_i \quad p = p_i \quad t_b = t_i \quad t_e = t_{i+1}$
for $(\,d,\, p_j,\, [t_j, t_{j+1})\,) \in I$ do
  $p'_{min} = min(\,p_{min},\, p_j\,) \quad p'_{max} = max(\,p_{max},\, p_j\,)$
  $p' = optrep(p'_{min}, p'_{max})$
  if $err_{rel}(p_{min},\, p') \le \epsilon \wedge err_{rel}(p_{max},\, p') \le \epsilon$ then
    $p_{min} = p'_{min} \quad p_{max} = p'_{max} \quad p = p' \quad t_e = t_{j+1}$
  else
    $O = O\, \cup\, (\,d,\, p,\, [t_b, t_e)\,)$
    $p_{min} = p_j \quad p_{max} = p_j \quad p = p_j \quad t_b = t_j \quad t_e = t_{j+1}$
  end if
end for
$O = O\, \cup\, (\,d,\, p,\, [t_b, t_e)\,)$
```

Algorithm 1 makes one pass over the input sequence I. While doing so, it coalesces sequences of postings having maximal length. The optimal representative for a sequence of postings depends only on their minimal and maximal payload (p_{min} and p_{max}) and can be looked up using *optrep* in $O(1)$ (see [16] for details). When reading the next posting, the algorithm tries to add it to the current sequence of postings. It computes the hypothetical new representative p' and checks whether it would retain the approximation guarantee. If this test fails, a coalesced posting bearing the old representative is added to the output sequence O and, following that, the bookkeeping is reinitialized. The time complexity of the algorithm is in $O(n)$.

Note that, since we make no assumptions about the sort order of index lists, temporal-coalescing algorithms have an additional preprocessing cost in $O(|L_v|\ log\, |L_v|)$ for sorting the index list and chopping it up into subsequences for each document.

6. SUBLIST MATERIALIZATION

Efficiency of processing a query $q^{\,t}$ on our time-travel inverted index is influenced adversely by the wasted I/O due to read but skipped postings. Temporal coalescing implicitly addresses this problem by reducing the overall index list size, but still a significant overhead remains. In this section, we tackle this problem by proposing the idea of *materializing sublists* each of which corresponds to a contiguous subinterval of time spanned by the full index. Each of these sublists contains all coalesced postings that overlap with the corresponding time interval of the sublist. Note that all those postings whose validity time-interval spans across the temporal boundaries of several sublists are replicated in each of the spanned sublists. Thus, in order to process the query $q^{\,t}$ it is sufficient to scan any materialized sublist whose time-interval contains t.

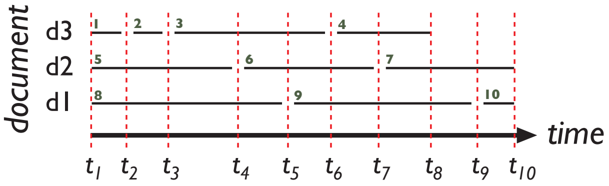


Figure 2: Sublist Materialization

We illustrate the idea of sublist materialization using an example shown in Figure 2. The index list L_v visualized in the figure contains a total of 10 postings from three documents d1, d2, and d3. For ease of description, we have numbered boundaries of validity time-intervals, in increasing time-order, as $t_1, \ldots, t_{10}$ and numbered the postings themselves as $1, \ldots, 10$. Now, consider the processing of a query $q^{\,t}$ with $t \in [t_1,\, t_2)$ using this inverted list. Although only three postings (postings 1, 5 and 8) are valid at time t, the whole inverted list has to be read in the worst case. Suppose that we split the time axis of the list at time t_2, forming two sublists with postings {1, 5, 8} and {2, 3, 4, 5, 6, 7, 8, 9, 10} respectively. Then, we can process the above query with optimal cost by reading only those postings that existed at this t.

At a first glance, it may seem counterintuitive to reduce index size in the first step (using temporal coalescing), and then to increase it again using the sublist materialization techniques presented in this section. However, we reiterate that our main objective is to improve the efficiency of processing queries, not to reduce the index size alone. The use of temporal coalescing improves the performance by reducing the index size, while the sublist materialization improves performance by judiciously replicating entries. Further, the two techniques, can be applied separately and are independent. If applied in conjunction, though, there is a synergetic effect – sublists that are materialized from a temporally coalesced index are generally smaller.

We employ the notation $L_v : [t_i,\, t_j)$ to refer to the materialized sublist for the time interval $[t_i,\, t_j)$, that is formally defined as,

$$L_v : [t_i,\, t_j) = \{(\,d,\, p,\, [t_b, t_e)\,) \in L_v \,|\, t_b < t_j \wedge t_e > t_i\}\ .$$

To aid the presentation in the rest of the paper, we first provide some definitions. Let $T = \langle\, t_1 \ldots t_n \,\rangle$ be the sorted sequence of all unique time-interval boundaries of an inverted list L_v. Then we define

$$\mathcal{E} = \{\, [t_i,\, t_{i+1}) \,|\, 1 \le i < n\}$$

to be the set of *elementary time intervals*. We refer to the set of time intervals for which sublists are materialized as

$$\mathcal{M} \subseteq \{\, [t_i,\, t_j) \,|\, 1 \le i < j \le n \,\}\ ,$$

and demand

$$\forall\, t \in [t_1,\, t_n)\ \exists\, m \in \mathcal{M} \,:\, t \in m\ ,$$

i.e., the time intervals in $\mathcal{M}$ must completely cover the time interval $[t_1,\, t_n)$, so that time-travel queries $q^{\,t}$ for all $t \in [t_1,\, t_n)$ can be processed. We also assume that intervals in $\mathcal{M}$ are disjoint. We can make this assumption without ruling out any optimal solution with regard to space or performance defined below. The *space* required for the materialization of sublists in a set $\mathcal{M}$ is defined as

$$S(\,\mathcal{M}\,) = \sum_{m \in \mathcal{M}} |L_v : m|\ ,$$

i.e., the total length of all lists in $\mathcal{M}$. Given a set $\mathcal{M}$, we let

$$\pi(\,[t_i,\, t_{i+1})\,) = [t_j,\, t_k) \,\in\, \mathcal{M} \,:\, [t_i,\, t_{i+1}) \,\subseteq\, [t_j,\, t_k)$$

denote the time interval that is used to process queries $q^{\,t}$ with $t \in [t_i,\, t_{i+1})$. The performance of processing queries $q^{\,t}$ for $t \in [t_i,\, t_{i+1})$ inversely depends on its *processing cost*

$$PC(\,[t_i,\, t_{i+1})\,) = |L_v : \pi(\,[t_i,\, t_{i+1})\,)| \;,$$

which is assumed to be proportional to the length of the list $L_v : \pi(\,[t_i,\, t_{i+1})\,)$. Thus, in order to optimize the performance of processing queries we minimize their processing costs.

6.1 Performance/Space-Optimal Approaches

One strategy to eliminate the problem of skipped postings is to eagerly materialize sublists for all elementary time intervals, i.e., to choose $\mathcal{M} = \mathcal{E}$. In doing so, for every query $q^{\,t}$ only postings valid at time t are read and thus the best possible performance is achieved. Therefore, we will refer to this approach as P_{opt} in the remainder. The initial approach described above that keeps only the full list L_v and thus picks $\mathcal{M} = \{\,[t_1,\, t_n)\,\}$ is referred to as S_{opt} in the remainder. This approach requires minimal space, since it keeps each posting exactly once.

P_{opt} and S_{opt} are extremes: the former provides the best possible performance but is not space-efficient, the latter requires minimal space but does not provide good performance. The two approaches presented in the rest of this section allow mutually trading off space and performance and can thus be thought of as means to explore the configuration spectrum between the P_{opt} and the S_{opt} approach.

6.2 Performance-Guarantee Approach

The P_{opt} approach clearly wastes a lot of space materializing many nearly-identical sublists. In the example illustrated in Figure 2 materialized sublists for $[t_1,\, t_2)$ and $[t_2,\, t_3)$ differ only by one posting. If the sublist for $[t_1,\, t_3)$ was materialized instead, one could save significant space while incurring only an overhead of one skipped posting for all $t \in [t_1,\, t_3)$. The technique presented next is driven by the idea that significant space savings over P_{opt} are achievable, if an upper-bounded loss on the performance can be tolerated, or to put it differently, if a performance guarantee relative to the optimum is to be retained. In detail, the technique, which we refer to as PG (Performance Guarantee) in the remainder, finds a set $\mathcal{M}$ that has minimal required space, but guarantees for any elementary time interval $[t_i, t_{i+1})$ (and thus for any query $q^{\,t}$ with $t \in [t_i,\, t_{i+1})$) that performance is worse than optimal by at most a factor of $\gamma \geq 1$. Formally, this problem can be stated as

$$\underset{\mathcal{M}}{argmin}\; S(\,\mathcal{M}\,) \quad \text{s.t.}$$

$$\forall\,[t_i,\, t_{i+1}) \,\in\, \mathcal{E} \,:\, PC(\,[t_i,\, t_{i+1})\,) \,\leq\, \gamma \cdot |L_v : [t_i,\, t_{i+1})| \;.$$

An optimal solution to the problem can be computed by means of *induction* using the recurrence

$$C(\,[t_1,\, t_{k+1})\,) =$$
$$min\,\{C(\,[t_1,\, t_j)\,) + |L_v : [t_j,\, t_{k+1})| \;|\; 1 \leq j \leq k \,\wedge\, condition\}\,,$$

where $C(\,[t_1,\, t_j)\,)$ is the optimal cost (i.e., the space required) for the prefix subproblem

$$\{\,[t_i,\, t_{i+1}) \,\in\, \mathcal{E} \,|\, [t_i,\, t_{i+1}) \,\subseteq\, [t_1,\, t_j)\,\}$$

and *condition* stands for

$$\begin{aligned}\forall\,[t_i,\, t_{i+1}) \in \mathcal{E} \,:\, [t_i,\, t_{i+1}) \subseteq [t_j,\, t_{k+1})\\ \Rightarrow |L_v : [t_j,\, t_{k+1})| \,\leq\, \gamma \cdot |L_v : [t_i,\, t_{i+1})|\end{aligned} \;.$$

Intuitively, the recurrence states that an optimal solution for $[t_1,\, t_{k+1})$ be combined from an optimal solution to a prefix subproblem $C(\,[t_1,\, t_j)\,)$ and a time interval $[t_j,\, t_{k+1})$ that can be materialized without violating the performance guarantee.

Pseudocode of the algorithm is omitted for space reasons, but can be found in the accompanying technical report [5]. The time complexity of the algorithm is in $O(n^{\,2})$ – for each prefix subproblem the above recurrence must be evaluated, which is possible in linear time if list sizes $|L : [t_i,\, t_j)|$ are precomputed. The space complexity is in $O(n^{\,2})$ – the cost of keeping the precomputed sublist lengths and memoizing optimal solutions to prefix subproblems.

6.3 Space-Bound Approach

So far we considered the problem of materializing sublists that give a guarantee on performance while requiring minimal space. In many situations, though, the storage space is at a premium and the aim would be to materialize a set of sublists that optimizes expected performance while not exceeding a given space limit. The technique presented next, which is named SB, tackles this very problem. The space restriction is modeled by means of a user-specified parameter $\kappa \geq 1$ that limits the maximum allowed blowup in index size from the space-optimal solution provided by S_{opt}. The SB technique seeks to find a set $\mathcal{M}$ that adheres to this space limit but minimizes the expected processing cost (and thus optimizes the expected performance). In the definition of the expected processing cost, $P(\,[t_i,\, t_{i+1})\,)$ denotes the probability of a query time-point being in $[t_i,\, t_{i+1})$. Formally, this space-bound sublist-materialization problem can be stated as

$$\underset{\mathcal{M}}{argmin} \sum_{[t_i,\, t_{i+1})\,\in\,\mathcal{E}} P(\,[t_i,\, t_{i+1})\,) \cdot PC(\,[t_i,\, t_{i+1})\,) \quad \text{s.t.}$$

$$\sum_{m \in \mathcal{M}} |L_v : m| \leq \kappa\, |L_v| \;.$$

The problem can be solved by using dynamic programming over an increasing number of time intervals: At each time interval in $\mathcal{E}$ the algorithms decides whether to start a new materialization time-interval, using the known best materialization decision from the previous time intervals, and keeping track of the required space consumption for materialization. A detailed description of the algorithm is omitted here, but can be found in the accompanying technical report [5]. Unfortunately, the algorithm has time complexity in $O(n^3|L_v|)$ and its space complexity is in $O(n^2|L_v|)$, which is not practical for large data sets.

We obtain an approximate solution to the problem using *simulated annealing* [22, 23]. Simulated annealing takes a fixed number R of rounds to explore the solution space. In each round a random successor of the current solution is looked at. If the successor does not adhere to the space limit, it is always rejected (i.e., the current solution is kept). A successor adhering to the space limit is always accepted if it achieves lower expected processing cost than the current solution. If it achieves higher expected processing cost, it is randomly accepted with probability $e^{\,-\Delta/r}$ where Δ is the increase in expected processing cost and $R \geq r \geq 1$ denotes the number of remaining rounds. In addition, throughout all rounds, the method keeps track of the best solution seen so far. The solution space for the problem at hand can be efficiently explored. As we argued above, we solely have to look at sets $\mathcal{M}$ that completely cover the time interval $[t_1, t_n)$ and do not contain overlapping time intervals. We represent such a set $\mathcal{M}$ as an array of n boolean variables $b_1 \ldots b_n$ that convey the boundaries of time intervals in the set. Note that b_1 and b_n are always set to "true". Initially, all $n-2$ intermediate variables assume "false", which corresponds to the set $\mathcal{M} = \{\, [t_1, t_n) \,\}$. A random successor can now be easily generated by switching the value of one of the $n-2$ intermediate variables. The time complexity of the method is in $O(n^2)$ – the expected processing cost must be computed in each round. Its space complexity is in $O(n)$ – for keeping the n boolean variables.

As a side remark note that for $\kappa = 1.0$ the SB method does not necessarily produce the solution that is obtained from S_{opt}, but may produce a solution that requires the same amount of space while achieving better expected performance.

7. EXPERIMENTAL EVALUATION

We conducted a comprehensive series of experiments on two real-world datasets to evaluate the techniques proposed in this paper.

7.1 Setup and Datasets

The techniques described in this paper were implemented in a prototype system using Java JDK 1.5. All experiments described below were run on a single SUN V40z machine having four AMD Opteron CPUs, 16GB RAM, a large network-attached RAID-5 disk array, and running Microsoft Windows Server 2003. All data and indexes are kept in an Oracle 10g database that runs on the same machine. For our experiments we used two different datasets.

The *English Wikipedia revision history* (referred to as WIKI in the remainder) is available for free download as a single XML file. This large dataset, totaling 0.7 TBytes, contains the full editing history of the English Wikipedia from January 2001 to December 2005 (the time of our download). We indexed all 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 892,255 documents with 13,976,915 versions having a mean (μ) of 15.67 versions per document at standard deviation (σ) of 59.18. We built a time-travel query workload using the query log temporarily made available recently by AOL Research as follows – we first extracted the 300 most frequent keyword queries that yielded a result click on a Wikipedia article (for e.g., "french revolution", "hurricane season 2005", "da vinci code" etc.). The thus extracted queries contained a total of 422 distinct terms. For each extracted query, we randomly picked a time point for each month covered by the dataset. This resulted in a total of $18,000$ $(= 300 \times 60)$ time-travel queries.

The second dataset used in our experiments was based on a subset of the European Archive [13], containing *weekly crawls of 11 .gov.uk websites throughout the years 2004 and 2005* amounting close to 2 TBytes of raw data. We filtered out documents not belonging to MIME-types `text/plain` and `text/html`, to obtain a dataset that totals 0.4 TBytes and which we refer to as UKGOV in rest of the paper. This included a total of 502,617 documents with 8,687,108 versions ($\mu = 17.28$ and $\sigma = 13.79$). We built a corresponding query workload as mentioned before, this time choosing keyword queries that led to a site in the `.gov.uk` domain (e.g., "minimum wage", "inheritance tax" , "citizenship ceremony dates" etc.), and randomly sampling a time point for every month within the two year period spanned by the dataset. Thus, we obtained a total of 7,200 ($= 300 \times 24$) time-travel queries for the UKGOV dataset. In total 522 terms appear in the extracted queries.

The collection statistics (i.e., N and $avdl$) and term statistics (i.e., DF) were computed at monthly granularity for both datasets.

7.2 Impact of Temporal Coalescing

Our first set of experiments is aimed at evaluating the approximate temporal coalescing technique, described in Section 5, in terms of index-size reduction and its effect on the result quality. For both the WIKI and UKGOV datasets, we compare temporally coalesced indexes for different values of the error threshold ϵ computed using Algorithm 1 with the non-coalesced index as a baseline.

	WIKI		**UKGOV**	
ϵ	**# Postings**	**Ratio**	**# Postings**	**Ratio**
-	8,647,996,223	100.00%	7,888,560,482	100.00%
0.00	7,769,776,831	89.84%	2,926,731,708	37.10%
0.01	1,616,014,825	18.69%	744,438,831	9.44%
0.05	556,204,068	6.43%	259,947,199	3.30%
0.10	379,962,802	4.39%	187,387,342	2.38%
0.25	252,581,230	2.92%	158,107,198	2.00%
0.50	203,269,464	2.35%	155,434,617	1.97%

Table 1: Index sizes for non-coalesced index (-) and coalesced indexes for different values of ϵ

Table 1 summarizes the index sizes measured as the total number of postings. As these results demonstrate, approximate temporal coalescing is highly effective in reducing index size. Even a small threshold value, e.g. $\epsilon = 0.01$, has a considerable effect by reducing the index size almost by an order of magnitude. Note that on the UKGOV dataset, even accurate coalescing ($\epsilon = 0$) manages to reduce the index size to less than 38% of the original size. Index size continues to reduce on both datasets, as we increase the value of ϵ.

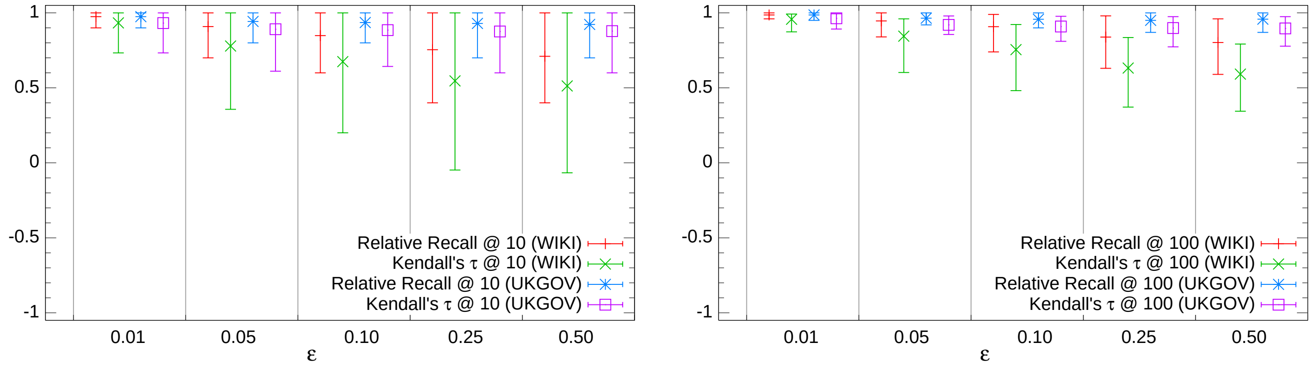
How does the reduction in index size affect the query results? In order to evaluate this aspect, we compared the top-k results computed using a coalesced index against the ground-truth result obtained from the original index, for different cutoff levels k. Let G_k and C_k be the top-k documents from the ground-truth result and from the coalesced index respectively. We used the following two measures for comparison: (i) *Relative Recall* at cutoff level k ($RR@k$), that measures the overlap between G_k and C_k, which ranges in [0, 1] and is defined as

$$RR@k = |G_k \cap C_k| / k \; .$$

(ii) *Kendall's* τ (see [7, 14] for a detailed definition) at cutoff level k ($KT@k$), measuring the agreement between two results in the relative order of items in $G_k \cap C_k$, with value 1 (or -1) indicating total agreement (or disagreement).

Figure 3 plots, for cutoff levels 10 and 100, the mean of $RR@k$ and $KT@k$ along with 5% and 95% percentiles, for different values of the threshold ϵ starting from 0.01. Note that for $\epsilon = 0$, results coincide with those obtained by the original index, and hence are omitted from the graph.

It is reassuring to see from these results that approximate temporal coalescing induces minimal disruption to the query results, since $RR@k$ and $KT@k$ are within reasonable limits. For $\epsilon = 0.01$, the smallest value of ϵ in our experiments, $RR@100$ for WIKI is 0.98 indicating that the results are



(a) @10

(b) @100

Figure 3: Relative recall and Kendall's τ observed on coalesced indexes for different values of ϵ

almost indistinguishable from those obtained through the original index. Even the relative order of these common results is quite high, as the mean KT@100 is close to 0.95. For the extreme value of $\epsilon = 0.5$, which results in an index size of just 2.35% of the original, the RR@100 and KT@100 are about 0.8 and 0.6 respectively. On the relatively less dynamic UKGOV dataset (as can be seen from the σ values above), results were even better, with high values of RR and KT seen throughout the spectrum of ϵ values for both cutoff values.

7.3 Sublist Materialization

We now turn our attention towards evaluating the sublist materialization techniques introduced in Section 6. For both datasets, we started with the coalesced index produced by a moderate threshold setting of $\epsilon = 0.10$. In order to reduce the computational effort, boundaries of elementary time intervals were rounded to day granularity before computing the sublist materializations. However, note that the postings in the materialized sublists still retain their original timestamps. For a comparative evaluation of the four approaches – P_{opt}, S_{opt}, PG, and SB – we measure space and performance as follows. The required space $S(\mathcal{M})$, as defined earlier, is equal to the total number of postings in the materialized sublists. To assess performance we compute the expected processing cost (EPC) for all terms in the respective query workload assuming a uniform probability distribution among query time-points. We report the mean EPC, as well as the 5%- and 95%-percentile. In other words, the mean EPC reflects the expected length of the index list (in terms of index postings) that needs to be scanned for a random time point and a random term from the query workload.

The S_{opt} and P_{opt} approaches are, by their definition, parameter-free. For the PG approach, we varied its parameter γ, which limits the maximal performance degradation, between 1.0 and 3.0. Analogously, for the SB approach the parameter κ, as an upper-bound on the allowed space blowup, was varied between 1.0 and 3.0. Solutions for the SB approach were obtained running simulated annealing for $R = 50,000$ rounds.

Table 2 lists the obtained space and performance figures. Note that EPC values are smaller on WIKI than on UKGOV, since terms in the query workload employed for WIKI are relatively rarer in the corpus. Based on the depicted results, we make the following key observations. i) As expected, P_{opt} achieves optimal performance at the cost of an enormous space consumption. S_{opt}, to the contrary, while consuming an optimal amount of space, provides only poor expected processing cost. The PG and SB methods, for different values of their respective parameter, produce solutions whose space and performance lie in between the extremes that P_{opt} and S_{opt} represent. ii) For the PG method we see that for an acceptable performance degradation of only 10% (i.e., $\gamma = 1.10$) the required space drops by more than one order of magnitude in comparison to P_{opt} on both datasets. iii) The SB approach achieves close-to-optimal performance on both datasets, if allowed to consume at most three times the optimal amount of space (i.e., $\kappa = 3.0$), which on our datasets still corresponds to a space reduction over P_{opt} by more than one order of magnitude.

We also measured wall-clock times on a sample of the queries with results indicating improvements in execution time by up to a factor of 12.

8. CONCLUSIONS

In this work we have developed an efficient solution for time-travel text search over temporally versioned document collections. Experiments on two real-world datasets showed that a combination of the proposed techniques can reduce index size by up to an order of magnitude while achieving nearly optimal performance and highly accurate results.

The present work opens up many interesting questions for future research, e.g.: How can we even further improve performance by applying (and possibly extending) encoding, compression, and skipping techniques [35]?. How can we extend the approach for queries $q^{[t_b, t_e]}$ specifying a time interval instead of a time point? How can the described time-travel text search functionality enable or speed up text mining along the time axis (e.g., tracking sentiment changes in customer opinions)?

9. ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their valuable comments – in particular to the reviewer who pointed out the opportunity for algorithmic improvements in Section 5 and Section 6.2.

10. REFERENCES

[1] V. N. Anh and A. Moffat. Pruned Query Evaluation Using Pre-Computed Impacts. In *SIGIR*, 2006.

[2] V. N. Anh and A. Moffat. Pruning Strategies for Mixed-Mode Querying. In *CIKM*, 2006.

	WIKI				UKGOV			
	S($\mathcal{M}$)	**EPC**			**S**($\mathcal{M}$)	**EPC**		
		5%	Mean	95%		5%	Mean	95%
P_{opt}	54,821,634,137	11.22	3,132.29	15,658.42	21,372,607,052	39.93	15,593.60	66,938.86
S_{opt}	379,962,802	114.05	30,186.52	149,820.1	187,387,342	63.15	22,852.67	102,923.85
$PG \quad \gamma = 1.10$	**3,814,444,654**	**11.30**	**3,306.71**	**16,512.88**	**1,155,833,516**	**40.66**	**16,105.61**	**71,134.99**
$PG \quad \gamma = 1.25$	1,827,163,576	12.37	3,629.05	18,120.86	649,884,260	43.62	17,059.47	75,749.00
$PG \quad \gamma = 1.50$	1,121,661,751	13.96	4,128.03	20,558.60	436,578,665	46.68	18,379.69	78,115.89
$PG \quad \gamma = 1.75$	878,959,582	15.48	4,560.99	22,476.77	345,422,898	51.26	19,150.06	82,028.48
$PG \quad \gamma = 2.00$	744,381,287	16.79	4,992.53	24,637.62	306,944,062	51.48	19,499.78	87,136.31
$PG \quad \gamma = 2.50$	614,258,576	18.28	5,801.66	28,849.02	269,178,107	53.36	20,279.62	87,897.95
$PG \quad \gamma = 3.00$	552,796,130	21.04	6,485.44	32,361.93	247,666,812	55.95	20,800.35	89,591.94
$SB \quad \kappa = 1.10$	412,383,387	38.97	12,723.68	60,350.60	194,287,671	63.09	22,574.54	102,208.58
$SB \quad \kappa = 1.25$	467,537,173	26.87	9,011.81	45,119.08	204,454,800	57.42	22,036.39	95,337.33
$SB \quad \kappa = 1.50$	557,341,140	19.84	6,699.36	32,810.85	246,323,383	53.24	20,566.68	91,691.38
$SB \quad \kappa = 1.75$	647,187,522	16.59	5,769.40	28,272.89	296,345,976	49.56	19,065.99	84,377.44
$SB \quad \kappa = 2.00$	737,819,354	15.86	5,358.99	27,112.01	336,445,773	47.58	18,569.08	81,386.02
$SB \quad \kappa = 2.50$	916,308,766	13.99	4,639.77	23,037.59	427,122,038	44.89	17,153.94	74,449.28
$SB \quad \kappa = 3.00$	**1,094,973,140**	**13.01**	**4,343.72**	**22,708.37**	**511,470,192**	**42.15**	**16,772.65**	**72,307.43**

Table 2: Required space and expected processing cost (in # postings) observed on coalesced indexes ($\epsilon = 0.10$)

[3] P. G. Anick and R. A. Flynn. Versioning a Full-Text Information Retrieval System. In *SIGIR*, 1992.

[4] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[5] K. Berberich, S. Bedathur, T. Neumann, and G. Weikum. A Time Machine for Text search. Technical Report MPI-I-2007-5-002, Max-Planck Institute for Informatics, 2007.

[6] M. H. Böhlen, R. T. Snodgrass, and M. D. Soo. Coalescing in Temporal Databases. In *VLDB*, 1996.

[7] P. Boldi, M. Santini, and S. Vigna. Do Your Worst to Make the Best: Paradoxical Effects in PageRank Incremental Computations. In *WAW*, 2004.

[8] 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*, 2006.

[9] C. Buckley and A. F. Lewit. Optimization of Inverted Vector Searches. In *SIGIR*, 1985.

[10] M. Burrows and A. L. Hisgen. Method and Apparatus for Generating and Searching Range-Based Index of Word Locations. U.S. Patent 5,915,251, 1999.

[11] S. Büttcher and C. L. A. Clarke. A Document-Centric Approach to Static Index Pruning in Text Retrieval Systems. In *CIKM*, 2006.

[12] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. S. Maarek, and A. Soffer. Static Index Pruning for Information Retrieval Systems. In *SIGIR*, 2001.

[13] http://www.europarchive.org.

[14] R. Fagin, R. Kumar, and D. Sivakumar. Comparing Top k Lists. *SIAM J. Discrete Math.*, 17(1):134–160, 2003.

[15] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[16] S. Guha, K. Shim, and J. Woo. REHIST: Relative Error Histogram Construction Algorithms. In *VLDB*, 2004.

[17] M. Hersovici, R. Lempel, and S. Yogev. Efficient Indexing of Versioned Document Sequences. In *ECIR*, 2007.

[18] http://www.archive.org.

[19] Y. E. Ioannidis and V. Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In *SIGMOD*, 1995.

[20] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantees. In *VLDB*, 1998.

[21] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An Online Algorithm for Segmenting Time Series. In *ICDM*, 2001.

[22] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[23] J. Kleinberg and E. Tardos. *Algorithm Design.* Addison-Wesley, 2005.

[24] U. Manber. *Introduction to Algorithms: A Creative Approach.* Addison-Wesley, 1989.

[25] K. Nørvåg and A. O. N. Nybø. DyST: Dynamic and Scalable Temporal Text Indexing. In *TIME*, 2006.

[26] J. M. Ponte and W. B. Croft. A Language Modeling Approach to Information Retrieval. In *SIGIR*, 1998.

[27] S. E. Robertson and S. Walker. Okapi/Keenbow at TREC-8. In *TREC*, 1999.

[28] B. Salzberg and V. J. Tsotras. Comparison of Access Methods for Time-Evolving Data. *ACM Comput. Surv.*, 31(2):158–221, 1999.

[29] M. Stack. Full Text Search of Web Archive Collections. In *IWAW*, 2006.

[30] E. Terzi and P. Tsaparas. Efficient Algorithms for Sequence Segmentation. In *SIAM-DM*, 2006.

[31] M. Theobald, G. Weikum, and R. Schenkel. Top-k Query Evaluation with Probabilistic Guarantees. In *VLDB*, 2004.

[32] http://www.wikipedia.org.

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

[34] J. Zhang and T. Suel. Efficient Search in Large Textual Collections with Redundancy. In *WWW*, 2007.

[35] J. Zobel and A. Moffat. Inverted Files for Text Search Engines. *ACM Comput. Surv.*, 38(2):6, 2006.