Efficient Time-Travel on Versioned Text Collections

Klaus Berberich, Srikanta Bedathur, Gerhard Weikum

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
{kberberi, bedathur, weikum}@mpi-inf.mpg.de

Abstract: The availability of versioned text collections such as the Internet Archive opens up opportunities for time-aware exploration of their contents. In this paper, we propose *time-travel retrieval and ranking* that extends traditional keyword queries with a temporal context in which the query should be evaluated. More precisely, the query is evaluated over all states of the collection that existed during the temporal context.

In order to support these queries, we make key contributions in (i) defining extensions to well-known relevance models that take into account the temporal context of the query and the version history of documents, (ii) designing an *immortal index* over the full versioned text collection that avoids a blowup in index size, and (iii) making the popular NRA algorithm for top-k query processing aware of the temporal context. We present preliminary experimental analysis over the English Wikipedia revision history showing that the proposed techniques are both effective and efficient.

1 Introduction

Numerous versioned text collections are available today, starting from Wikis where textual content is explicitly version-controlled, to the World Wide Web whose dynamic content is only partly archived through efforts such as the Internet Archive [IA]. Similarly, information feeds of Web portals, Weblogs, etc., with their time-stamping mechanism can also be regarded as versioned text collections.

Despite the ubiquity of such versioned text collections, querying and ranking over them is typically only done on the most recent snapshot of the collection. Although this suffices for many simple information needs, certain information needs cannot be adequately answered. Consider the following examples:

1. Finding historical statutes is one of the common information needs of attorneys [Coh]. With almost all public documents of the state now available on the Web, it is natural to use the historical Web to unearth these documents rather than to wade through libraries. Ideally, a query “Indiana court rules” and a time-window of interest should suffice to retrieve relevant documents and thus to speed up legal research.

2. In a case of copyright-infringement a lawyer needs to collect evidence about the illegal copying of copyrighted material (e.g., a piece of writing) on the Web. The

suspect has removed the copied material in the meanwhile. Ideally, a query would consist of a snippet of the copyrighted material and the time-window during which the copied material was available.

3. For discovering a population affected by a disease, say HIV, a public health administrator could query a database of electronic health records [ISO] containing longitudinal record of health history of individuals, with keywords relating to symptoms of the disease – e.g., "swollen lymph nodes", "sore throat" etc., and a time-window of interest – say, in the last 3 years, and expect to be presented a list of candidate records that should be tested in detail.

Similar situations where keyword querying needs to be combined with a time-window of interest also arise for other kinds of business data such as customer-support databases, CRM databases, patent records, etc. Today's search engines cannot provide adequate results for such *temporally contextualized* information needs. The relevant (versions of) Web pages have disappeared and are therefore not contained in the current snapshot that is considered. Though Web archives may contain these relevant versions, they provide only limited access functionality such as the Wayback Machine [IA] requiring the knowledge of URLs or extensive browsing within the archived contents.

The idea of querying time-varying relational databases has been well-studied in temporal database research [TCG$^+$93]. However, there has been very little work in translating those ideas into the context of versioned text collections where keyword querying, not SQL, is the query interface. Further, unlike in standard relational databases, the results of textual querying need to be ranked taking into account the relevance to the given keywords.

1.1 Time-travel in Text Search

In this paper, we introduce the notion of *time-travel querying and ranking* aimed at supporting temporally contextualized information needs such as those mentioned above. Informally, our time-travel query model adds a *time-window of interest* to the standard keyword-based query model. The given keyword query should be *evaluated and ranked* over the state(s) of the text collection valid during the given time-window.

A straightforward approach to supporting these time-travel queries would be to periodically generate a new, updated index, and retain the old index. Whenever a query specifies a time-window, consult the indexes that are valid for that time-window and answer the query. Although this approach is easy to implement, it clearly entails a complete replication of the index even when many documents have undergone none or only minor changes. Even if one may reduce the index size with a variety of compression techniques, it is not clear how to score and rank the documents (based on their versions) when the time-window of interest spans more than one index instance. A document could be relevant with high score in one index, while in the subsequent index it could no longer be relevant or could have a different relevance score based on content changes in the meanwhile. The relevance score of a document could vary not only due to local changes within the document, but also due to collection-wide changes that affected the eliteness of a query term. It is not clear at a glance how to reconcile these variations in score to generate a meaningful document-level result ranking for the time-window of interest.

The key technical challenges in supporting time-travel queries over large versioned text collections are:

1. **Indexing:** As we pointed out above, if each version of the document in the text collection is independently indexed, many entries are replicated unnecessarily. It is essential to develop an indexing strategy that eliminates this redundancy and controls the associated space explosion.
2. **Ranking model:** Commonly used ranking models such as tf-idf and Okapi BM25 are only applicable to static text collections. However, in the case of time-travel queries, the ranking model has to generate a document-level ranking that takes into account the variations in score of individual versions that are valid within the time-window of interest.
3. **Query processing:** Users are typically interested in quickly obtaining few top results to the given time-travel query. Therefore, it is important to develop efficient query processing techniques to identify such top-k answers, in a spirit similar to the popular family of Threshold Algorithms [FLN03].

1.2 Contributions

In this paper, we explore in detail the issues surrounding the explicit incorporation of a temporal context in text retrieval. We introduce and formally define time-travel querying as a mechanism for keyword-based searching over versioned text collections. In order to efficiently answer these queries, we develop an immortal text indexing technique. In contrast to prior work in searching on versioned text [AF92, NN06], our approach ensures fairness in access costs *irrespective* of the age of relevant documents and the position of the time-window of interest. Finally, we propose a novel time-contextualized ranking framework that goes well beyond the simple boolean querying supported in [NN06].

In summary, the key contributions made in this paper are the following:

1. A *relevance model for versioned document collections* is introduced. We consider the document-level result granularity, with each document being a sequence of its associated versions. This is in contrast to the version-level granularity treated in earlier research.
2. We present an indexing system called I2T2 (Immortal Index for Time-Travel) that enriches the traditional inverted file index with temporal information and allows for result ranking. We decouple contributions from local (term-document-specific) and global (term-collection-specific) statistics to the relevance score, and utilize this to reduce the index size.

3. In order to further overcome the index-size blowup on account of negligible changes between versions, we introduce *approximate temporal coalescing* of index entries. We empirically show that this can achieve significant reductions in index size while retaining good accuracy.

4. Addressing the issue of query processing over our extended index, we present TC-NRA (Time Context aware NRA), an adaptation of the NRA algorithm proposed by Fagin et al. [FLN03], to generate top-k answers efficiently for time-travel queries.

5. Finally, we present a comprehensive experimental evaluation of the proposed techniques over the English Wikipedia revision history.

1.3 Organization

The remainder of the paper is organized as follows. Section 2 puts our work in context of related work. We introduce our data and query model, and then present our relevance model in Section 3. Following that, in Section 4, we describe the I2T2 indexing system. The top-k query processing algorithm TC-NRA is presented next in Section 5. The proposed techniques are comprehensively evaluated in a series of experiments that is detailed in Section 6. Finally, in Section 7, we conclude the present work and outline future directions of research.

2 Related Work

There is only scarce work that deals specifically with *temporal text indexing* – the main problem considered here. Anick and Flynn [AF92], as a first notable exception, describe a help-desk system that enriches inverted file indexes to allow for retrieval of historical document versions. The system supports historical queries both for a given point in time or time span. Access costs are optimized for accesses to the most recent versions and increase the farther one moves into the past, thus making the approach inappropriate for the scenario considered in this work. Nørvåg and Nybø [NN06] concentrate on text-containment queries only and do not consider the ranked retrieval case.

Textual querying is typified by the need for effective ranking of results and has been the focus of research for more than 30 years. The tf-idf scoring model is traditionally used a as measure of a document's relevance to the given query keywords. This scoring model combines the term frequency (tf) and the logarithmically dampened inverse document frequency (idf) of the term. The individual per-term scores are aggregated to generate the final document score. A more sophisticated Probabilistic IR based model, Okapi BM25 [RW94, RW99] has become prevalent recently as it addresses many of the shortcomings of the simple tf-idf model. Okapi BM25 provides a smoothed non-linear influence of all ranking components in the final score. It normalizes term frequencies taking into account document lengths. It has been shown to be superior to the tf-idf model in many retrieval tasks.

One form of time-aware querying that is very common is the search for news articles [CGR05]. The main challenge in searching news is to automatically incorporate publication times of news articles in order to generate "interesting" rankings. However, there is no notion of versioning for news article, i.e., each occurrence is considered as an explicit document instance, and the rankings are generated accordingly. In contrast, our work aims at allowing an explicit time-window of interest in queries, and providing document-level rankings for the specified time-window.

Inverted file indexes are an efficient way to index text and are deployed in numerous of today's systems including major Web search-engines. For a recent comprehensive overview of inverted file indexes we refer to Zobel and Moffat [ZM06]. The main attractions of inverted lists include their excellent compressibility, spatial locality of reference, and adaptability to a wide variety of query interfaces. Maintenance of inverted file indexes against evolving text collections has been studied in the contexts of document additions/deletions, as well as, in-place updates to documents during incremental crawling [BCL06]. However, the goal of all these previous proposals is to keep the index up-to-date with the contents of the text collection – no effort is made in capturing the evolution of the collection in the index. Interestingly, there have been proposals such as [SGM97] that have addressed the issue of capturing the history of database evolution, but their solution is simply to maintain separate, periodically updated indexes for a time window in the past. In contrast, the techniques presented here aim at capturing the history of the collection in a *single immortal index*, while retaining most of the high-performance features of standard inverted file indexes.

There has been more than two decades of work on incorporating the temporal evolution of databases into index structures (for an excellent survey we refer to [ST99]). However, most of the techniques proposed in this context solve only the boolean aspect of the query, e.g., retrieve the tuple(s) that *satisfy* the given constraint. Querying versioned text collections as we consider them in this paper necessitates an entirely different querying model (keyword-based querying) and requires a ranking of results. In addition, while the retrieval granularity in the case of temporal databases is at the record-version level, here we focus on aggregated document-level result granularity in order to provide effective results.

Temporal coalescing, a technique that we adapt and extend in the scope of this work, was originally proposed by Böhlen et al. [BSS96] in the context of temporal databases.

3 Data and Query Model

We consider a general model of a dynamic document collection where each *document* refers to a collection of *versions*. The textual content is held in each version, and the inter-document references also point to individual versions rather than to the corresponding document.

Formally, a document $\mathbf{d}$ is modeled as a sequence of timestamped versions d^{t_i}, i.e.,

$$\mathbf{d} = \langle d^{t_i}, d^{t_{i+1}}, \ldots \rangle \, .$$

The timestamps associated with versions represent either the creation times or observation times of the version, depending on the application. Each version d^{t_i} can be considered as a vector of terms as in the vector-space model. Document versions are assumed valid until replaced by a newer version of the same document or deleted from the collection. The document deletions are modeled by replacing the document vector by a special zero vector, $\perp$, that denotes an *empty* and *inactive* document. A document version d^{t_i} has the implicit validity time-interval $[t_i, t_{i+1})$ with t_{i+1} being set to now for the current version. The set of timestamps of versions valid for a given time-period $[t_b, t_e]$ is denoted as $V(\mathbf{d}, [t_b, t_e])$, i.e.,

$$V(\mathbf{d}, [t_b, t_e]) = \{t_i \mid d^{t_i} \in \mathbf{d} \, \wedge \, d^{t_i} \neq \perp \, \wedge t_i \leq t_e \, \wedge \, t_{i+1} > t_b\} \, .$$

We use the notation d^t to refer to the version of d valid at time t.

We formulate time-varying forms of document-specific and collection-specific statistics. Thus, let $tf(v, d^t)$ denote term frequency (i.e, the number of occurrences) of term v in the document version d^t. The length of document version d^t is referred to as $dl(d^t)$. As collection-specific statistical values we define $N(t)$ as the collection size at time t and $avdl(t)$ as the average length of documents, both computed over only the versions that are valid at time t. Apart from that, let $df(v, t)$ denote the number of documents in the collection that contain term v at time t.

Using the above model, it is possible to represent not only the scenario where documents are explicitly version-managed – such as in Wikis, but also when versions are inferred implicitly like in the case of collections obtained through repeated crawls of the Web. Note that in the latter case, it is possible to have missed a few changes to a document as a result of the crawl strategy, but we consider this to be an issue orthogonal to our setting. Further, the model naturally allows for deletion and subsequent reappearance of documents.

3.1 Query Model

A *time-travel query* $q^{[t_b, t_e]}$ consists of a *content part* q and a *temporal context* $[t_b, t_e]$. The content of a query has the same structure as a version in our data model, and can be regarded as a set of keywords. For the evaluation of the query only versions that exist at any point in the time interval of interest $[t_b, t_e]$ are taken into account. If a time-travel query is specified without a temporal context we consider that to be equivalent to the case when the temporal context spans the complete life-time of the archive.

3.1.1 Version-level Relevance Scoring

Next, we adapt two existing relevance models, namely Okapi BM25 and a tf-idf model, to quantify the relevance of a document version d^{t_i} to a query $q^{[t_b, t_e]}$ as a score $w(q^{[t_b, t_e]}, d^{t_i})$. Both models require slight adaption, since statistical values such as document frequencies *df* and the collection size $avdl$ are fixed in their original formulations and not time-varying as in our scenario.

TF-IDF Scoring The tf-idf model considered here combines term frequencies with logarithmically dampened inverse document frequencies. The inverse document frequency of a term v at time t is defined as

$$idf(v,t) = log\,\frac{N(t)}{1 + df(v,t)}$$

and, as stated above, is time-varying in our setting. For a temporal context $[t_b, t_e]$ we therefore aggregate inverse document frequencies in the respective time-window and let $\overline{idf}(v, [t_b, t_e])$ denote the aggregated value. In this paper, we consider $\overline{idf}(v, [t_b, t_e])$ obtained by a simple averaging of idf values in the time-window, although other forms of aggregation are also applicable. Altogether we yield the following formulation for the tf-idf model

$$w(q^{[t_b,t_e]}, d^{t_i}) = \sum_{v \in q} tf(v, d^{t_i}) \cdot \overline{idf}(v, [t_b, t_e]) \tag{1}$$

Okapi BM25 Okapi BM25 [RW99] employs a different definition of inverse document frequencies as

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

and extends term frequencies taking into account document lengths. Again, we aggregate inverse document frequencies over the temporal context as $\overline{idf}(v, [t_b, t_e])$, and obtain

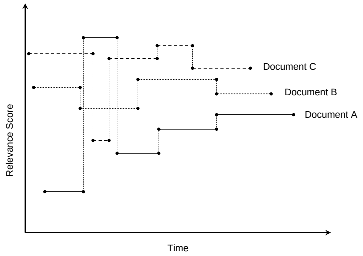
$$w(q^{[t_b,t_e]}, d^{t_i}) = \sum_{v \in q} \frac{(k_1 + 1) \cdot tf(v, d^{t_i})}{k_1 \cdot ((1-b) + b \cdot \frac{dl(d^{t_i})}{avdl(t_i)}) + tf(v, d^{t_i})} \cdot \overline{idf}(v, [t_b, t_e)) \tag{2}$$

as our adaptation of Okapi BM25. The parameters k_1 and b are inherited from the original model and are commonly set to values 1.2 and 0.75 respectively.

3.2 Document-level Score Aggregation

With the model introduced so far we can assess the relevance of a document version to a given $q^{[t_b,t_e]}$. In order to obtain the document-level relevance, we need to combine the version-level relevance model at a coarser granularity. To this end, we aggregate relevance scores $w(q^{[t_b,t_e]}, d^{t_i})$ of document versions d^{t_i} that existed at any point in the temporal context $[t_b, t_e]$ to obtain a *document relevance score* $w(q^{[t_b,t_e]}, \mathbf{d})$. We consider three such document relevance models, which differ in the way version relevance scores are aggregated. Figure 1 illustrates the rankings under these models.

MIN: This relevance model conservatively judges the relevance of a document based on the least relevant of its versions. Thus, for a document to score high under this



Aggregation Model	Ranking
MIN	**B**, **C**, **A**
MAX	**A**, **C**, **B**
TAVG	**C**, **B**, **A**

Figure 1: Document Level Score Aggregations

model, all of its versions must achieve high relevance. The document relevance is defined as

$$w(q^{[t_b,t_e]}, \mathbf{d}) = min\{w(q^{[t_b,t_e]}, d^t) \,|\, t \in V(\mathbf{d}, [t_b, t_e])\} \;. \tag{3}$$

MAX: In contrast, the MAX relevance model takes the most relevant version as an indicator of document relevance. Under this model, a document may score high if at least one of its versions achieves high relevance. Formally the document relevance is then defined as

$$w(q^{[t_b,t_e]}, \mathbf{d}) = max\{w(q^{[t_b,t_e]}, d^t) \,|\, t \in V(\mathbf{d}, [t_b, t_e])\} \;. \tag{4}$$

TAVG: Finally, the TAVG relevance model considers the relevance of all versions to assess document relevance. Obviously, a simple sum of version relevance scores would favor frequently changing documents. Instead, we use the mean version relevance in $[t_b, t_e]$, which takes into account the lifespan of each version and is defined as

$$w(q^{[t_b,t_e]}, \mathbf{d}) = \begin{cases} \frac{1}{t_e - t_b} \int_{t_b}^{t_e} w(q^{[t_b,t_e]}, d^t)\, dt & \text{if } t_b \neq t_e \\ w(q^{[t,t]}, d^t) & \text{if } t_b = t_e = t \end{cases} \tag{5}$$

Since the function $w(q^{[t_b,t_e]}, d^t)$ is piecewise-constant in time, the integral in the above equation can be efficiently computed as a weighted summation of these segments. Unlike the MIN and MAX models the TAVG model is more robust to outliers in version relevance scores, considering the entirety of version relevance scores. The MIN (MAX) relevance model, on the other hand, may assign low (high) relevance to a document if one of its versions achieves a (low) high relevance score.

We illustrate the effect of these aggregation models over three documents – **A**, **B**, and **C**, whose relevance scores vary with time as illustrated in Figure 1. Under the MIN model, we see that document **B** is ranked above the other two documents, since the least relevant version of **B** is better than the least relevant versions of either **A** or **C** – although the other two documents have at least one version whose relevance score is more than any version of **B**. On the other hand, under the MAX model, document **A** comes on top of results, as its most relevant version is clearly better than that of the other two documents. Finally, the TAVG captures the notion of "consistency" of a document (in terms of its relevance score). Thus, the document **C** is given higher preference since almost all of its versions have consistently good relevance score – except for one single version that existed only during a very small time-window.

Later, in Section 5, we show that all three document relevance models can be efficiently implemented using the index structure I2T2 (Immortal Index for Time-Travel) which we propose in the next section.

4 Temporal Text Indexing

In this section we introduce the design of our temporal text-indexing system I2T2. We first describe our temporal extensions to the inverted file index and, following that, present ways to reduce index size using temporal coalescing.

4.1 Index Design

In order to provide the querying functionality outlined in the previous section, we need to efficiently compute the version-relevance scores $w(q^{[t_b,t_e]}, d^{t_i})$ and their document-level aggregations.

Both the tf-idf model and the Okapi BM25 model for version relevance considered above are summations of products of a term-document-specific measure and a term-collection-specific measure (i.e., the first and second factor within the summation of Equation 1 and 2 respectively). In our design, the term-document-specific tf-component and term-collection-specific idf-component are maintained separately, which is in contrast to common practice for static document collections where only their product is kept. In our setup, however, this separation adds the flexibility of computing the idf-component depending on the temporal context of the query. Thus the I2T2 system consists of the following three components:

1. *Versions*: the sequence of version timestamps per document
2. *IDF*: the time series of idf-scores per term
3. *TF*: the time series of tf-scores per occurring term per document.

The version index which lists the version timestamps of every document is employed for the computation of validity time-interval for each version of a document. As mentioned earlier, a version is considered to be valid until another version is detected, or, the document has disappeared from the collection. This information is organized as a B-Tree.

As individual documents in the collection undergo changes over time, the overall collection-level term statistics also vary as an aggregated effect. According to the relevance models presented earlier in Section 3.1, given a temporal context $[t_b, t_e]$ we need to efficiently compute the corresponding aggregated idf-score of the term. We maintain for each term the sequence of its idf-scores along with the associated validity time-interval. These idf-scores are computed collection-wide for all terms at regular intervals by taking into account only the active documents. At query time we scan this idf-score sequence for all query terms and retrieve the entries that are valid for the specified query time-window.

Most crucial for the efficiency of query processing, however, is the way per-term per-document information is kept, since, as we detail below, accessing this pool contributes most of the query-processing cost.

The per-term per-document information is typically maintained in an inverted file index. In brief, an inverted file index consists of a *vocabulary* that maps each term to its *inverted list*. The vocabulary is commonly organized as a B-Tree to efficiently lookup the inverted list for a given term. The *inverted list* for a term is a list of *postings* that are tuples containing a document identifier and additional payload like the frequency of the term in the document. Depending on the query task, these lists are sorted differently – e.g., in document-identifier, or term-frequency order. For a detailed recent survey about inverted file indexes we refer to Zobel and Moffat [ZM06].

Our extension to the inverted file index affects: (i) the structure of postings and (ii) the order of inverted lists. We build our extended inverted file index over all versions of documents, and record the information about the life of versions explicitly in the index. As a result, each posting now includes the validity time-interval of the corresponding document version. In other words, the posting

$$(d, tf, t_i, t_j)$$

in the inverted list belonging to term v expresses that for term v, document d had score tf during the time interval $[t_i, t_j)$. We thus obtain one such posting (d, tf, t_i, t_{i+1}) for each term v occurring in a document version d^{t_i}.

Furthermore, for reasons that become clear in the following sections, we keep inverted lists sorted with descending score as the primary sort order, ascending document identifiers as the secondary sort order, and increasing left boundaries of validity time-intervals as tertiary sort order.

4.2 Index-size Reduction through Temporal Coalescing

In our extended inverted file index described above, it is easy to observe that a lot of space is wasted by having one posting per term that occurs in a document version. In the evolution of a typical document over time, the changes are not very drastic, thus the associated tf-values of most terms is unchanged between two adjacent versions. We now introduce *temporal coalescing* as a means to exploit this feature of document evolution and to counter the waste of space. The idea behind temporal coalescing is to coalesce temporally adjacent postings belonging to the same document that have identical payloads. We can further reduce the index size by performing *approximate temporal coalescing* – where two postings are coalesced if their payloads are similar to each other. In the rest of this section, we describe these index reduction strategies.

Accurate Temporal Coalescing

Accurate temporal coalescing coalesces temporally adjacent postings only if their payloads, i.e. tf values, are identical. Thus, two postings, (d, tf_0, t_0, t_1) and (d, tf_1, t_1, t_2), are coalesced into (d, tf_0, t_0, t_2) only if $tf_0 = tf_1$. We can implement accurate temporal coalescing efficiently as a single pass algorithm presented in Algorithm 1, by exploiting the sort-order of our index lists.

Algorithm 1 Accurate Temporal Coalescing

```
$input = \langle(d_i, tf_i, t_{b_i}, t_{e_i})\rangle$
$output = \emptyset$
$b = null$
for $(d_i, tf_i, t_{b_i}, t_{e_i}) \in input$ do
    if $b \neq null \wedge (b.d = d_i \wedge b.t_e = t_{b_i} \wedge b.tf = tf_i)$ then
        $b.t_e = t_{e_i}$
    else
        $b = (d_i, tf_i, t_{b_i}, t_{e_i})$
        $output.append(b)$
    end if
end for
```

Algorithm 1 scans the non-coalesced input inverted list in sorted order. The result of the current round of coalescing is maintained in a posting buffer lt. For each posting read from the inverted list, the algorithm either coalesces the posting with lt, or reinitializes lt by copying the contents of the posting just read, after appending lt to the output inverted list.

Accurate temporal coalescing reduces the index size without affecting the resulting rankings. However, the scale of this space reduction clearly depends on the likelihood of having identical payloads in temporally adjacent postings. For the tf-idf relevance model (Equation 1), where the payloads are simply the term frequencies, this likelihood is high as typical changes to a document leave large portions of the document unchanged. In contrast, under the Okapi BM25 relevance model, as can be seen from Equation 2, payloads of temporally adjacent postings are less likely to be identical. There are two reasons for this: (i) a single change in the document that affects the document length, e.g., adding a few words, is likely to make the payloads of all new postings different from those of their temporally preceding postings, and (ii) the collection-level average document length measure could change even when there are no changes to the document, also affecting the payloads of postings.

Approximate Temporal Coalescing

In order to overcome the limitations of accurate temporal coalescing, we introduce *approximate temporal coalescing*. This is based on our hypothesis that temporally adjacent postings can be coalesced even if their payloads differ to a small degree without inducing much error in the final query result. Thus, temporally adjacent postings are coalesced even if their payloads differ, as long as the error made through the use of coalesced posting is within a tunable bound.

Formally, for an input sequence of temporally adjacent postings I belonging to document d we aim to find a shorter sequence O of coalesced postings so that

$$\begin{aligned} &\forall (d, tf_i, t_i, t_{i+1}) \in I \,:\, \forall (d, tf_j, t_j, t_{j+1}) \in O \,: \\ &\qquad t_j \leq t_i \,\wedge\, t_{i+1} \leq t_{j+1} \Rightarrow error(tf_j, tf_i) \leq \epsilon \;. \end{aligned}$$

In the formula above, $error$ is a measure of the approximation error made by using the payload tf' of the coalesced posting, and ϵ is a tunable threshold value the error must not exceed. For $error$, we use the relative error measure, defined as

$$error_{rel}(tf', tf) = |tf' - tf| \,/\, max\{c, |tf|\} \;,$$

where c is a sanity constant, which we fix at 10^{-4}.

The problem above can be mapped to finding a piecewise-constant representation for the N points (t_i, tf_i) retaining our guarantee on the error. Finding piecewise-constant representations is a well-studied problem in time-series segmentation [KCHP01, TT06] and histogram construction [GSW04, IP95, JKM$^+$98]. Using dynamic programming an optimal solution having a minimal number of B^* postings (segments, buckets) can be found in time $O(N^2 B^*)$ [JKM$^+$98]. The greedy approximate Algorithm 2 has $O(N)$ time complexity and performs well in practice. It produces solutions that retain the guarantee on the error and have a close-to-optimal number of postings.

We note that the algorithm, in contrast to Algorithm 1, takes as an input a document-specific sequence of temporally adjacent postings sorted in ascending order of the left validity time-interval-boundary. For a sequence of temporally adjacent postings the auxiliary function $coalesce$ produces a single posting minimizing the maximum $error$. For the relative error measure that we use, this optimal value can be looked up as described in Guha et al. [GSW04], thus computable in constant time. In each step, Algorithm 2 attempts to add the newly read posting to the buffer b, which holds a sequence of postings that can be coalesced while retaining the error guarantee. If the attempt fails, i.e., the coalesced posting produced by $coalesce$ does not retain the error guarantee, the coalesced posting corresponding to the contents of the buffer b is added to the $output$ and the buffer b is reinitialized.

Algorithm 2 Approximate Temporal Coalescing

```
$input = \langle (d, tf_i, t_{b_i}, t_{e_i}) \rangle$
$output = \emptyset$
$b = \emptyset$
**for** $(d, tf_i, t_{b_i}, t_{e_i}) \in I$ **do**
    **if** $b = \emptyset$ **then**
        $b.append((d, tf_i, t_{b_i}, t_{e_i}))$
    **else**
        $lt = coalesce(b \cup (d, tf_i, t_{b_i}, t_{e_i}))$
        **if** $\exists (d, tf_j, t_{b_j}, t_{e_j}) \in (b \cup (d, tf_i, t_{b_i}, t_{e_i})) \, : \, error(tf_j, lt.tf) > \epsilon$ **then**
            $output.append(coalesce(b))$
            $b = \langle (d, tf_i, t_{b_i}, t_{e_i}) \rangle$
        **else**
            $b.append((d, tf_i, t_{b_i}, t_{e_i}))$
        **end if**
    **end if**
**end for**
$output.append(coalesce(b))$
```

5 Query Processing for Ranked Retrieval

Having introduced our index design and temporal coalescing as a way to reduce index size, we now discuss how queries $q^{[t_b, t_e]}$ can be efficiently evaluated under the different relevance models introduced in Section 3. Since we are interested in the ranked retrieval of top results for a query, we adapt the popular No Random Accesses (NRA) algorithm from the family of threshold algorithms [FLN03] for query processing. We coin our extended algorithm TC-NRA (Temporal Context aware NRA).

5.1 Preliminaries: NRA Algorithm

Before we describe our TC-NRA algorithm, we briefly recall some details about the original NRA. As the name indicates, NRA determines the top-k result using only cheap sorted accesses to the inverted lists in a round-robin fashion, and avoids expensive random accesses. At each step of the algorithm, the candidate documents are kept in a main-memory pool. For each document seen in at least one of the lists, the set of lists on which the document score has been evaluated is maintained. Based on the scores seen so far in the inverted lists, for each candidate a $worstscore$ and a $bestscore$ value is maintained. These reflect its worst and best possible scores, respectively, including the lists in which the document is yet unseen. The minimal $worstscore$ in the current top-k, referred to as min_k, acts as a threshold for stopping the index scans. The algorithm terminates when none of the candidates has a $bestscore$ value exceeding min_k. When this is the case, none of the remaining candidates can anymore make it into the top-k. The algorithm is guaranteed to produce the correct top-k result if $worstscore$s ($bestscore$s) increase (decrease) monotonically.

5.2 TC-NRA Algorithm

The NRA algorithm described above is clearly oblivious to the notion of document versions (and their corresponding validity time-intervals), as well as, to the time-window of interest expressed in the query. In order to adapt the NRA algorithm to our setting, we extend it in the following two key aspects: (i) the bookkeeping associated with candidates during the run of the algorithm, and (ii) the definitions of $worstscore$ and $bestscore$ of a candidate. The resulting algorithm, called TC-NRA, is presented in Algorithm 3.

Recall that the required result granularity for time-travel queries is at the level of documents as collections of versions. Referring to equations 3, 4, and 5, we see that the relevance score of a document in the query temporal-context $[t_b, t_e]$ depends on the scores of its versions that are valid during this period. Thus, unlike in the case of original NRA where keeping one $worstscore$ and $bestscore$ per candidate is sufficient, we now have to maintain one $worstscore$ and $bestscore$ for each document version that existed in the temporal context $[t_b, t_e]$. We maintain these as two time series, denoted as $worstscores$ and $bestscores$. Next, the bookkeeping of evaluated terms per candidate is extended – for a dimension (term) v the time intervals for which the dimension has been evaluated already are kept track of in $evaluated[v]$. In the pseudocode of Algorithm 3 we make use of some notational shortcuts. When advancing the cursor on the current list (line 10), the method $next([t_b, t_e])$ advances the cursor to the next posting (d, s, t_k, t_l) that exists during the temporal context, i.e., fulfills $t_k < t_e \wedge t_l \geq t_b$. Further, we use abbreviated notation for updating the $worstscores$ (line 15) and $bestscores$ (line 19), thus when writing $worstscores[T] = worstscores[T] + s$ the value s is added to $worstscores[t]$ for all $t \in T$.

We move on to the definition of $worstscore$ (line 15) and $bestscore$ (line 34) of a document, as an aggregation of the respective values of its constituent versions. Clearly, the model employed for document-level aggregation also affects the definitions of these values. Accordingly, we present appropriate definitions that satisfy the monotonicity requirements of NRA, under each of the three document-level score-aggregation models we have considered.

MIN For the MIN relevance model, $worstscore$ and $bestscore$ are defined as

$$\begin{aligned} worstscore(d) &= min\{\, d.worstscores[t] \mid t \in [t_b, t_e] \,\} \\ bestscore(d) &= min\{\, d.bestscores[t] \mid t \in [t_b, t_e] \,\} \ . \end{aligned}$$

Algorithm 3 Time Context aware No Random Accesses (TC-NRA) Algorithm

```
$min_k = 0$; $candidates = \emptyset$; $topk = \emptyset$

/*   fetch index lists for terms $v$ in query $q$   */
$L = \{\, v \mid v \in q \,\}$

**for** $v \in L$ **do**
    $(d, s, t_k, t_l) = v.next(\, [t_b, t_e]\, )$
    $high_v = s$

    /*   do bookkeeping   */
    $T = [t_k, t_l) \setminus d.evaluated[v]$
    $d.worstscores[T] = d.worstscores[T] + s$
    $d.evaluated[v] = d.evaluated[v] \cup T$
    **for** $w \in L$ **do**
        $T = [t_k, t_l) \setminus d.evaluated[w]$
        $d.bestscores[T] = d.worstscores[T] + high_w$
    **end for**

    /*   manage candidates and topk   */
    **if** $worstscore(d) > min_k$ **then**
        $topk.removeMinimum()$
        $topk.insert(d)$
        $candidates.remove(d)$
    **else if** $bestscore(d) > min_k$ **then**
        $candidates.insert(d)$
    **else**
        $candidates.remove(d)$
    **end if**
    $min_k = min\{\, worstscores(d) \mid d \in topk\}$

    /*  check stopping criterion  */
    **if** $candidates = \emptyset \vee max\{\, bestscore(d') \mid d' \in candidates \,\} \leq min_k$ **then**
        return $topk$
    **end if**
**end for**
```

MAX For the MAX relevance model we define $worstscore$ and $bestscore$ as

$$\begin{aligned} worstscore(d) &= max\{\, d.worstscores[t] \mid t \in [t_b, t_e]\,\} \\ bestscore(d) &= max\{\, d.bestscores[t] \mid t \in [t_b, t_e]\,\}\ . \end{aligned}$$

TAVG For the TAVG relevance model we define $worstscore$ and $bestscore$ as

$$\begin{aligned} worstscore(d) &= \begin{cases} \frac{1}{t_e - t_b} \int_{t_b}^{t_e} d.worstscores[t]\, dt & : \text{ if } t_b \neq t_e \\ d.worstscores[t] & : \text{ if } t_b = t_e = t \end{cases} \\ bestscore(d) &= \begin{cases} \frac{1}{t_e - t_b} \int_{t_b}^{t_e} d.bestscores[t]\, dt & : \text{ if } t_b \neq t_e \\ d.bestscores[t] & : \text{ if } t_b = t_e = t \end{cases} \ . \end{aligned}$$

Monotonicity of $worstscore$ and $bestscore$ in all three cases follows directly from the monotonicity of each individual $worstscore[t]$ ($bestscore[t]$) that can only increase (decrease) in the course of Algorithm 3.

6 Experiments

The presented techniques were implemented in a prototype system using Java JDK 1.5. All experiments were run on a SUN V40z machine having four AMD Opteron CPUs, 16 GB RAM, and a large network-attached RAID-5 disk array. All data is kept in an Oracle 10g database that runs on the same machine.

6.1 Setup & Datasets

As a dataset for our experiments we used the *English Wikipedia revision history* that is available for download as one large XML file. This dataset contains the editing history of the English Wikipedia spanning the time-window from January 2001 to December 2005 (the time of our download). This rich dataset contains much more information than the mere encyclopedia articles, among them, for instance, discussions associated with articles. We refer to Buriol et al. [BDLM05] for a detailed discussion of the dataset and its properties. For our experiments, we randomly picked 20% of versions, excluding those versions marked as minor revisions (e.g., if only typos are corrected). This yielded a total of 892,255 documents and 2,795,383 versions, with average of 3.13 versions per document at sample standard deviation 14.24. We computed idf-scores for each term on this dataset at monthly intervals and maintained it as the idf time-series component of I2T2.

We extracted a set of 45 queries from a large query log that was recently made available by AOL research, however, is no longer publicly available for download. The selected queries are the most frequent queries among those that led to a result click on a Wikipedia article. For each query we generated six temporal contexts. Three temporal contexts were chosen as the years 2003, 2004, and 2005. Further, for each query and each year we randomly picked one temporal context that coincides with one quarter, thus giving us another three temporal contexts per query.

6.2 Index Sizes

We computed indices both for the tf-idf and the Okapi BM25 model using approximate coalescing varying the threshold ϵ as $0\%, 1\%, 5\%, 10\%$, yielding a total of eight different indices. Note that $\epsilon = 0\%$ corresponds to accurate coalescing and is considered as the baseline in our experiments. We computed the top-100 for each keyword query, each temporal context, each index, and each document-level score-aggregation model resulting in a total of 6,480 queries that were processed. The resulting index sizes (measured as the total number of postings) are given in Table 1. The non-coalesced indexes for tf-idf and Okapi BM25, which we computed for comparison, have a total number of $1,244,168,879$ postings.

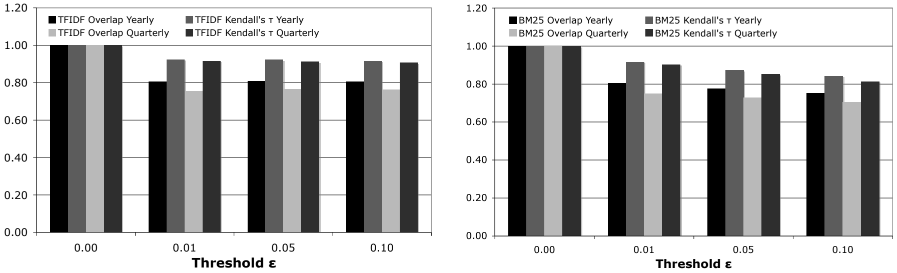
	tf-idf		Okapi BM25	
Threshold	Number of Postings	Ratio	Number of Postings	Ratio
0%	1,129,472,027	1.0000	1,186,996,012	1.0000
1%	180,196,069	0.1595	595,180,256	0.5014
5%	178,581,509	0.1581	327,687,360	0.2761
10%	177,003,930	0.1567	245,488,770	0.2068

Table 1: Index Sizes

The figures show that even for a small threshold $\epsilon = 1\%$ the index under the tf-idf model is reduced to about 16% of its original size. Similarly, for the Okapi BM25 model index size steadily decreases as ϵ is increased. For $\epsilon = 10\%$, as an example, the size of the index resulting from approximate coalescing is about 21% the size of the original index.

6.3 Accuracy

In the first experiment, we analyze how much the top-100 changes as we increase the threshold ϵ. To this end, for every query, we compare the top-100 obtained for a particular value of ϵ against the top-100 produced by the baseline. Two result lists are compared using the following two measures: The *overlap* is the fraction of documents that occurs in both result lists and does not take into account the order of documents. *Kendall's τ* is computed for the documents that appear in both results lists and reflects the agreement of the two lists on the order of documents. Here, a value of 1 (-1) indicates perfect agreement (disagreement) in order. Figure 2(a) and Figure 2(b) give mean overlap and mean Kendall's τ values over all queries and temporal contexts for tf-idf and Okapi BM25 relevance models respectively.



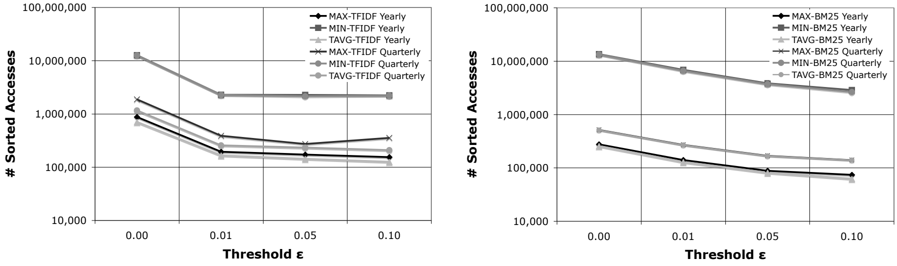
(a) TF-IDF (b) Okapi BM25

Figure 2: Accuracy@top-100

We consistently observe that *overlap* and Kendall's τ decrease gracefully as we increase the threshold ϵ for both the tf-idf and the Okapi BM25 relevance model. Further, we observe larger *overlap* and Kendall's τ values for the tf-idf relevance model. Combining these results with the index sizes reported in Table 1, we see that for $\epsilon = 10\%$ and the tf-idf relevance model, for instance, on an index that is only 16% the size of the original index, we yield a top-100 list containing on average 75 documents that are also present in the original result list.

6.4 Performance

Next, we analyze the performance of our approach for different values of the threshold ϵ and across the different document-level score-aggregation models that were introduced in Section 3. For both tf-idf and Okapi BM25 in combination with yearly and quarterly temporal contexts, we run our set of queries as one batch for MAX, MIN, and TAVG. The total number of sorted accesses that is needed to process this batch of queries is used as a measure of performance. The resulting figures for tf-idf and Okapi BM25 models are given in Figure 3(a) and Figure 3(b).



(a) TF-IDF (b) Okapi BM25

Figure 3: Performance@top-100

A first observation that can be made is that the MIN model is clearly the most expensive requiring up to one order of magnitude more sorted accesses. The MAX and TAVG models, in contrast, require about the same significantly lower numbers of sorted accesses. For all curves we observe that, as we increase the threshold ϵ from 0% to 10%, the number of required sorted accesses reduces by nearly an order of magnitude. While we observe a steady decrease in the number of sorted accesses for Okapi BM25, the respective figures for tf-idf exhibit a sharp drop for $\epsilon = 1\%$ that is followed by almost no further decrease. We note that this is in accordance with the index sizes reported in Table 1 that exhibit analogous behavior.

7 Conclusions

In this paper, we have explored in detail the issues in supporting a novel breed of temporally contextualized queries, called time-travel queries, over versioned document collections. We introduced a model for versioned document collections, and developed three relevance scoring models that aggregate version-level scores at a document-level granularity. In order to efficiently evaluate time-travel queries under the proposed relevance models, we designed an immortal indexing system called I2T2 and an efficient top-k processing algorithm called TC-NRA. Our experiments over the English Wikipedia revision history showed that our indexing strategy can be tuned to reduce the index size by close to an order of magnitude, while retaining Kendall's τ value close to 0.8 and about 75% content overlap in the top-100. We believe that these preliminary results show significant promise for supporting time-travel querying functionality and also opens avenues for a number of directions of future research. We plan to continue these studies over a wide variety of datasets including Web collections such as WebBase and Internet Archive. We are also investigating a variety of optimizations for the I2T2 system both in terms of its space-consumption and efficient index management. Another interesting direction we plan to explore is to automatically relax the temporal context in time-travel queries, thus treating it rather as a preference parameter of the user than as a hard condition for filtering results. Our final goal is to integrate these features into a popular, open-source text indexing system for web archives such as NutchWAX [Nut].

References

[AF92] Peter G. Anick and Rex A. Flynn. Versioning a Full-text Information Retrieval System. In *Proc. of ACM-SIGIR*, 1992.

[BCL06] Stefan Büttcher, Charles L. A. Clarke, and Brad Lushman. Hybrid index maintenance for growing text collections. In *Proc. of ACM-SIGIR*, 2006.

[BDLM05] Luciana Salete Buriol, Debora Donato, Stefano Leonardi, and Stefano Millozzi. Link and Temporal Analysis of Wikigraphs. Technical report, Department of Computer and System Sciences, University of Rome ”La Sapienza”, 2005.

[BSS96] Michael H. Böhlen, Richard Thomas Snodgrass, and Michael D. Soo. Coalescing in Temporal Databases. In *Proc. of VLDB*, 1996.

[CGR05] Gianna M. Del Corso, Antonio Gulli, and Francesco Romani. Ranking a stream of news. In *Proc. of WWW*, 2005.

[Coh] Steven M. Cohen. Remember the Internet Archive for Historical Research. http://www.llrx.com/features/internetarchive.htm.

[FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[GSW04] Sudipto Guha, Kyuseok Shim, and Jungchul Woo. REHIST: Relative Error Histogram Construction Algorithms. In *Proc. of VLDB*, 2004.

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

[IP95] Yannis E. Ioannidis and Viswanath Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In *Proc. of ACM SIGMOD*, 1995.

[ISO] Health informatics – Electronic health record – Definition, scope and context. ISO Technical Report ISO/TR 20514.

[JKM$^+$98] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal Histograms with Quality Guarantees. In *Proc. of VLDB*, 1998.

[KCHP01] Eamonn J. Keogh, Selina Chu, David Hart, and Michael J. Pazzani. An Online Algorithm for Segmenting Time series. In *Proc. of ICDM*, 2001.

[NN06] Kjetil Nørvåg and Albert Overskeid N Nybø. DyST: Dynamic and Scalable Temporal Text Indexing. *Proc. of TIME*, 2006.

[Nut] NutchWAX. http://archive-access.sourceforge.net/projects/nutch.

[RW94] Stephen E. Robertson and Steve Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *Proc. of ACM SIGIR*, 1994.

[RW99] Stephen E. Robertson and Steve Walker. Okapi/Keenbow at TREC-8. In *Proc. of TREC*, 1999.

[SGM97] Narayanan Shivakumar and Hector Garcia-Molina. Wave-indices: indexing evolving databases. In *Proc. of ACM SIGMOD*, 1997.

[ST99] Betty Salzberg and Vassilis J. Tsotras. Comparison of access methods for time-evolving data. *ACM Comput. Surv.*, 31(2):158–221, 1999.

[TCG$^+$93] Abdullah Uz Tansel, James Clifford, Shashi K. Gadia, Sushil Jajodia, Arie Segev, and Richard T. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation.* Benjamin/Cummings, 1993.

[TT06] Evimaria Terzi and Panayiotis Tsaparas. Efficient Algorithms for Sequence Segmentation. In *Proc. of SIAM Data Mining Conference*, 2006.

[ZM06] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.