Temporal-Textual Retrieval: Time and Keyword Search in Web Documents

Ali Khodaei, Cyrus Shahabi
Computer Science Department, University of Southern California, USA
and
Amir Khodaei
Electrical Engineering and Computer Sciences Deptt., University of California, Berkeley, USA

As the web ages, many web documents become relevant only to certain time periods, such as web-pages containing news and events or those documenting natural phenomena. Hence, to retrieve the most relevant pages, in addition to providing the relevant keywords, one may desire to identify the relevant time period(s) as well, e.g., "`Barack Obama 1980-1985`". Unfortunately, not much work has been done by industry or academia to support this type of searches. To the best of our knowledge, the only way that some search engines exploit the time information in the user query is to filter out those resulting web pages whose publication/modification time are not within the queried time interval. In this paper, we propose a new indexing and ranking framework for temporal-textual retrieval. The framework leverages the classical vector space model and provides a complete scheme for indexing, query processing and ranking of the temporal-textual queries. We propose a variety of approaches to exploit popular keyword and temporal index structures. We present a novel hybrid index structure which indexes both the temporal and the textual aspects of the documents in a unified, integrated manner. We also study how to rank documents by seamlessly combining their temporal and textual features. We develop a new scoring schema called temporal tf-idf to compute the temporal relevance of a document to a query, and we combine this score with the textual relevance to compute the overall relevance score of the document to the query. We present both a cost model analysis and an extensive set of experiments over real-world datasets (New York Times Annotated Corpus and Freebase) to evaluate the proposed framework and demonstrate its efficiency and effectiveness.

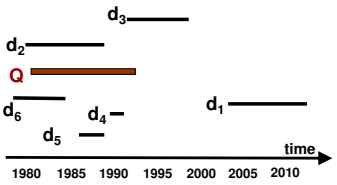
Keywords: Web Search, Time-aware ranking, Indexing, Temporal information retrieval

1. INTRODUCTION

For the first time in human history, there is a medium, the World Wide Web, that is continuously documenting our lives as they happen. Hence, it is such a waste to only be able to search this rich history by keywords and not by time. The web is no longer a snapshot of our history, neither should its search scheme. Unfortunately, the content of web-pages is currently not time-tagged, but this will become common practice in the near future (the same way that pages are now being geo-tagged) and until then, many techniques for automatic extraction of temporal information from documents [Verhagen *et al.* 2009], [Wong *et al.* 2005], [Alonso *et al.* 2007] will serve the purpose. Consequently, the challenge is how to enable efficient search of time-tagged web-pages? Or even more fundamentally, how to rank the results of a keyword-time search? Should a page with more textual similarity to the query keywords get ranked higher or the one with less textual similarity but higher temporal similarity? What does temporal similarity or relevance even mean and how it should be quantified? This paper is our initial attempt to address some of these fundamental and exciting open problems.

Several search engines have already started to exploit time in their search process. For instance, Google has started to add a feature called *search result option* that allows users to filter their search results by a custom time interval. For these search engines, the time attribute is usually the

This research is supported in part by Award No. 2011-IJ-CX-K054 from National Institute of Justice, Office of Justice Programs, U.S. Department of Justice, as well as the NSF grant IIS-1115153, the USC Integrated Media Systems Center (IMSC), and also by unrestricted cash gifts from Google, Microsoft and NGC. The opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsors such as NSF or NIJ.



(a)

Title	Time
Iraq Invasion	2003-2010
Iran-Iraq War	1980-1988
Kurdish Civil War	1991-1997
Gulf War	1990-1991
Al-Anfal Campaign	1986-1989
Reagan Presidency	1981-1985

Document	Keywords	
	Iraq	war
d_1	12	4
d_2	10	9
d_3	5	4
d_4	11	6
d_5	7	0
d_6	0	0

(b)

A tempo-textual query on documents with time information.

publication time (or the last modified time) of a document. The main assumption here is that the time-tag is a single time point, which simplifies the temporal "retrieval" problem to the temporal "filter" problem. That is, the final relevance (and ranking) of a document is still determined by the document's textual relevance to the query keywords, and the temporal feature of the document is only used to filter the documents in or out of the final result set. To illustrate why this is a simpler problem, consider the analogous situation where each web document contains a single keyword. In this case, for each web document, we simply need to check whether its keyword exists in the list of query keywords or not. There would be no need for relevance metrics such as tf-idf or indexing techniques such as inverted files. Similarly, if we relax the assumption that the temporal aspect of a web document is represented by a single time point, but by one or more time points and/or intervals, then we need temporal similarity metrics and index structures to efficiently estimate the temporal relevance of a web document to one or more query time points/intervals. Note that given the sophistication of the web content, it is no longer feasible to represent the temporal content of many web documents with a single time point. For instance, a Wikipedia page describing the biography of an individual is very likely to have several references of time, each relating to an event/achievement during the person's lifetime. Even if we consider representing only the published and modified time of a web document (instead of its content time), a single time point representation is simplistic as the publication and/or modification of a web document evolves through multiple time intervals or time points.

This paper considers a new kind of top-k query that takes into account both the temporal information in a document's content and the textual keywords of the document. An example query may search for "`Lakers Celtics Rivalry between years 1984 and 1986`". We call this type of query a *Temporal-Textual Retrieval* ("tempo-textual" for short) query. The answer to a top-k tempo-textual query is a list of k documents ranked according to a scoring function that combines their textual and temporal *relevances* to query keywords and the query timestamp, respectively. The tempo-textual query is different from queries that retrieve textually relevant documents within a time range or queries that rank the relevant documents based only on their temporal feature.

Example 1: Consider a collection of web pages, each one describing an event using textual keywords and also including one (or more) time-intervals (e.g., days, month, years or even decades). Suppose Tom is a student researching the *history of war in Iraq between years 1982 to 1992*. He submits a query to the system with two textual keywords "`Iraq`" and "`war`" and specifies "`1982 - 1992`" as the temporal expression.

Suppose there exist six documents in our collection with temporal information close (in time) to the query's temporal expression. Figure 1a shows these documents' temporal information plus the query's temporal expression (for a better readability, time-intervals are shown at different levels). Figure 1b shows each document's title and the frequencies of the two query keywords in each document. Tom wants to find top-3 relevant documents to the query.

In this example, document d_6 is not a relevant document since it does not contain any of the query keywords. Document d_1 is not very relevant to the query either, since its time-interval is very far from the query's time-interval (interestingly, this document is very likely to be the most relevant result when a regular textual search is used). The other four documents are overlapping (in time) with query's time-interval and contain at least one of the query keywords, therefore all could be potentially of interest to the user. However, it is not clear how to measure the relevance for documents and rank them in accordance with each other. For instance, it is clear that document d_2 should have a high relevance since its time-interval is very similar (has a significant overlap) to the query's time-interval and also contains both query keywords. On the other hand, it is not clear how relevant document d_5 is to the query. Although its time-interval is contained in the query time-interval, it does not have one of the query keywords. The other two documents, d_3 and d_4 both have the two query keywords and both have overlaps with the query time-interval (with various periods of overlap).

In this paper, we first present a simple baseline indexing technique to answer the temporal-textual queries. The baseline approach uses only one textual index structure and calculates the temporal relevance on-the-fly. We also introduce three new hybrid index structures to index documents based on both the temporal and textual features of the documents. Using one textual index structure (inverted file) and one temporal index structure (interval-tree) we present three different variants of combining a textual index and a temporal index to answer temporal-textual queries. We argue that these techniques would not be efficient in all circumstances and their performance is highly dependent on the distribution/selectivity of the data (e.g., distribution of textual keywords and temporal intervals in the corpus) and type of queries issued.

Next, we introduce our novel tempo-textual retrieval framework based on the classical vector space model. Following the same intuitions and techniques used in textual searches and inspired by the tf-idf schema in textual context, we define a new scoring schema called *temporal tf-idf* for temporal context. Using textual and temporal tf-idf, we define a new tempo-textual relevance score and ranking.

The third contribution of this paper is a novel tempo-textual index structure. Designing an efficient index structure for both textual and temporal data has several challenges. First, text and time are two totally different data types requiring different index structures. An ideal index should be able to handle both the temporal and the textual data simultaneously and in an integrated fashion. Second, the meaning of temporal relevance and textual relevance and how to combine them into one aggregate relevance score using the index structure have to be defined accurately. Moreover, the index structure needs to support cases where the influences of text and time on the overall relevance are different. Third, the ranking and search processes should not be separated. Otherwise, the ranking process will rank all the candidate documents (instead of only the relevant documents), making the query processing inefficient. Last but not the least, it should be straightforward to integrate the proposed index structure into the existing search engines.

In this paper, we propose a new hybrid index structure called *Tempo-Textual Inverted Index* ("T^2I^2" for short) for efficient search and ranking of time and text in a unified manner. T^2I^2 is an inverted index capable of indexing and searching both textual and temporal data in a similar, integrated manner. Towards this end, the time domain is divided into a number of consecutive cells and each cell is treated similar to a textual keyword. We present the structure of T^2I^2, and discuss two efficient algorithms for answering tempo-textual queries using T^2I^2.

Overall, we present a complete framework of indexing, query processing and ranking, for answering tempo-textual queries. Using experimental evaluation, we show that the proposed framework is both efficient and accurate.

The remainder of this paper is organized as follows. In Section 2, we formally define the tempo-textual search problem. Section 3 presents our baseline approach for simple processing of the tempo-textual queries. In Section 4, we introduce three new hybrid index structures for

the processing of tempo-textual queries. In Section 5, we introduce a new ranking mechanism to calculate the temporal relevance and tempo-textual relevance scores. In Section 6, we present our efficient hybrid index structure and show how it is used in query processing. We discuss how the proposed framework can be extended to support more general scenarios in Section 7. Section 8 empirically evaluates our proposed solution in terms of effectiveness and performance. Section 9 presents the related work to our problem. Finally, Section 10 presents the conclusion.

2. PRELIMINARIES

2.1 Problem Definition

We assume a collection $D = \{d_0,d_1,...,d_n\}$ of n documents (web pages). Each document d is composed of a set of keywords K_d and a *timespan*[1] T_d represented by two timestamps: *begin time* denoted by t_s and *end time* denoted by t_e. Terms t_s and t_e represent the number of time units (e.g. days) from a reference point in time (which is the same for all the documents). The difference between t_e and t_s (in number of time units) is defined as *timespan length*. For a better readability, we will use a normal date format (e.g. November 7th, 1980) for t_s and t_e in our examples. We will use the *timespan* to refer to T_d in this paper.

Tempo-textual query: A tempo-textual query is defined as $Q = \langle K_q, T_q \rangle$, where T_q is the temporal part of query specified as one timespan and K_q is a set of keywords in the query.

Temporal relevance: Temporal relevance between a document d and the query q is defined based on the type of the temporal relationship that exists between T_d and T_q. We focus only on the *overlap* relationship, although our approach can easily be extended to cover other temporal relationships [Allen 1981]. Subsequently, we define temporal relevance as follows: A document d is temporally relevant to the query q if the query's timespan has a non-empty intersection with the document's timespan, i.e., $T_q \cap T_d \neq \emptyset$. The larger the area of the intersection is, the more temporally relevant d and q are. We denote temporal relevance of document d to query q by $tmRel_q(d)$.

Textual relevance: A document d is textually relevant to the query q if there exists at least one keyword belonging to both d and q, i.e., $K_q \cap K_d \neq \emptyset$. The more keywords q and d has in common, the more they are textually relevant. We represent textual relevance of document d to query q by $txRel_q(d)$. See Section 2.2 for more information regarding textual relevance.

Tempo-textual relevance: A document d is tempo-textual relevant to the query q if it is both temporally and textually relevant to the query q. Tempo-textual relevance can be defined by a monotonic scoring function F of textual and temporal relevance. For example, F can be the weighted sum of the temporal and textual relevances:

$$F_q(d) = \begin{cases} \alpha.tmRel_q(d) + (1-\alpha).txRel_q(d) & \text{if } tmRel_q(d) > 0 \\ & \text{and } txRel_q(d) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

α is a parameter assigning relative weights to temporal and textual relevance. The output of function $F_q(d)$ is the tempo-textual relevance score of document d to query q, and is denoted by $ttRel_q(d)$. In Section 5 we show in details how to calculate tempo-textual relevance using our proposed index.

Tempo-textual search: A tempo-textual search identifies all the documents (web pages) that are *tempo-textual* relevant to q. The result are the top-k documents sorted based on documents' tempo-textual relevance scores. The parameter k is determined by the user.

[1]Throughout our definitions and examples, for simplicity we consider one timespan for each document and one timespan for each query. In Section 7, we show how to generalize this model to multiple timespans.

2.2 Textual Relevance

2.2.1 *tf-idf Score.* All current textual (keyword) search engines use a *similarity measure* to rank and identify potential (textual) relevant documents. In most keyword queries, a similarity measure is determined by using the following important parameters:

(1) $f_{d,k}$: the frequency of keyword k in document d
(2) $\max(f_{d,k})$: maximum value of $f_{d,k}$ over all the keywords in document d
(3) $\overline{f_{d,k}}$: normalized $f_{d,k}$, which is $\frac{f_{d,k}}{\max(f_{d,k})}$
(4) f_k: the number of documents containing one or more occurrences of keyword k

Using these values, three monotonicity observations are enforced [Zobel *et al.* 2006]: (1) less weight is given to the terms that appear in many documents; (2) more weight is given to the terms that appear many times in a document; and (3) less weight is given to the documents that contain many terms. The first property is quantified by measuring the inverse of frequency of keyword k among the documents in the collection. This factor is called *inverse document frequency* or the *idf* score. The second property is quantified by the raw frequency of keyword k inside a document d. This is called *term frequency* or tf score, and it describes how well that keyword describes the contents of the document [Baeza-Yates *et al.* 1999]. The third property is quantified by measuring the total number of keywords in the document. This factor is called *document length.*

A simple and very common formula to calculate the similarity between a document d and the query q is shown in Equation 2.

$$\begin{aligned} w_{q,k} &= \ln(1+\frac{n}{f_k}); \qquad w_{d,k} = \ln(1+\overline{f_{d,k}}); \\ W_d &= \sqrt{\sum_k w_{d,k}^2}; \qquad W_q = \sqrt{\sum_k w_{q,k}^2}; \\ S_{q,d} &= \frac{\sum_k w_{d,k}.w_{q,k}}{W_d.W_q}. \end{aligned} \tag{2}$$

n is the total number of documents in the system. Variable $w_{d,k}$ captures *the tf score* while variable $w_{q,k}$ captures *the idf score.* W_d represents *document length* and W_q is query length (which can be neglected since it is a constant for a given query). Finally, $S_{q,d}$ is the similarity measure (cosine similarity between document and query vectors) showing how relevant document d and query q are. In this case (textual context) it is the same as $txRel_q(d)$.

3. BASELINE APPROACH

In this section, we briefly discuss our baseline index structure and algorithm that exploit existing techniques for processing tempo-textual queries.

IIO (Inverted Index Only): The basic idea behind IIO is to leverage the inverted index to calculate textual relevance using a classical tf-idf model of all the documents, therefore obtaining a ranked list of the documents based on their textual relevance. The list is then scanned to check if each (textually relevant) document's temporal expression has a non-empty overlap with the query's temporal expression. For each overlapping document d, we calculate the temporal relevance between d and the query q as a simple overlap function as follows:

$$tmRel_q(d) = \begin{cases} \frac{T_d \cap T_q}{T_q} & \text{if } (T_d \cap T_q) < T_q \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

where $\frac{T_d \cap T_q}{T_q}$ is the area of overlap between the document time interval T_d and the query time interval T_q divided by the area of T_q. After calculating the temporal relevance, we compute the

final relevance score as shown earlier in Equation 1 for the documents returned from the inverted index. We sort the results and return the top-k results to the user. Using this straightforward simple approach, we do not need to calculate the temporal relevance for all the documents but only for the documents with the textual relevance greater than 0. We can use the following optimization to improve the performance of the above approach. While scanning (sorted) documents from the inverted index, if the score of d is smaller than the score of the kth document in the intermediatory results, then we skip this document and go to the next one. Otherwise (or if we do not have k intermediatory results yet), we insert d into the list of k intermediatory results (in its proper position) and remove the kth element from the list. We stop scanning when we are sure that none of the existing documents can generate a score larger than the current kth document in the intermediatory result set (or when we reach the end of the list). The *IIO* algorithm only uses the inverted index and calculates the temporal relevance on-the-fly.

4. HYBRID APPROACHES

The baseline method described previously only makes use of a textual index structure (i.e., inverted file) and does not employ a temporal indexing structure. As a result, all the documents returned from the textual index (which may be a very large set) must be accessed and temporal relevance for most, if not all, of the documents must be computed. In this section, we propose three hybrid index structures, each using both temporal indexing and textual indexing thus enabling us to prune irrelevant documents both temporally and textually. As before, we use an inverted file as the textual index structure. For temporal indexing, we use an interval-tree [Preparata *et al.* 1985] which allows us to efficiently find all time intervals that overlap with a given (query) time interval. Interval tree is an ordered tree data structure to hold intervals. Specifically, it allows one to efficiently find all intervals that overlap with any given interval (or point). An interval tree for a set of m intervals uses $O(m)$ storage and has height $O(log\, m)$. It can be built in $O(m.log\, m)$ time. It takes $O(log\, m)$ time to process a query and return overlapping intervals. Further details regarding interval tree can be found in [Preparata *et al.* 1985].

In all index structures in this section, an interval-tree is built on documents' time intervals and then used to filter documents overlapping with the query's temporal expression. By adding a few simple steps to the search algorithm of the interval-tree, we also calculate the amount of overlap during the search process. Using the overlap duration, we calculate normalized temporal scores ($tmRel_q(d)$ in Equation1) for each document as follows: $tmRel_q(d) = overlapLength_q(d)/maxOverlapLength$ where $overlapLength_q(d)$ is the length of overlap between document d's and query q's temporal expressions and $maxOverlapLength$ is maximum of such values for q.

4.1 Inverted File and Interval-Tree Index (FnT)

This hybrid index structure combines the inverted file and interval-tree separately and in an independent fashion. In this structure, documents are indexed by both index structures separately. While all textual information is indexed by an inverted file (as is done in textual search engines), all the temporal information is indexed by an interval-tree. The primary difference between the interval-tree used here and traditional interval-trees is that, here each leaf node of the interval-tree points to a list of documents containing the time-interval corresponding to that leaf node.

With this structure, when processing query q two independent processes take place. First, the textual part of the query K_q is fed into the inverted file and the textual relevance for each document is calculated. Second, the temporal part of the query T_q is sent to the interval-tree and temporal intervals overlapping with T_q are found. Subsequently, for each overlapping time-interval, the temporal relevance between each time-interval and T_q is computed. Finally, lists for each leaf node (time-interval) are accessed and corresponding temporal relevance values are assigned to the documents in each list. After finishing above steps, two sets of lists are merged, final relevance scores are computed and top-k results are returned.

4.2 Inverted File Then Interval-Tree Index (FtT)

With the *inverted file then interval-tree* index structure, (vocabulary of) an inverted file is constructed on top of all textual keywords. However, instead of pointing to inverted lists, each keyword in the vocabulary points to (the root node of) an interval tree. Interval tree for keyword k_i is built on temporal-intervals existed in documents containing k_i. Each leaf node of each interval-tree points to a (page) list of documents that 1) their time-interval overlaps with the time-interval corresponding to that leaf node, and 2) contain the textual keyword pointing to that interval-tree. In other words, we get a set of (page) lists whose entry is determined by a pair of a keyword and a time-interval. We call a pair of a keyword and a time-interval *time-interval-keyword (TIK)* if there is document which contains the textual keyword and whose temporal expression (time-interval) overlaps with the time-interval.

For query processing, textual part of the query K_q is read first. For each keyword in the query, corresponding interval-tree is accessed and processed until all leaf nodes (time-intervals) overlapping with the temporal part of the query are found. For each time-interval, the temporal relevance is calculated and saved. Next, (page) lists for each leaf node (time-interval) are traversed. While traversing page lists, textual relevance of each document is also computed. Finally, all the page lists (from all TIKs) are merged and temporal and textual relevances are combined.

4.3 Interval-Tree Then Inverted File Index (TtF)

With the *interval-tree then inverted file* index structure, first an interval-tree is constructed on top of all the time-intervals in the system. Instead of pointing to a list, each leaf node of the tree points to an inverted file. An inverted file for each time-interval (leaf node) is built on top of all documents overlapping with that time-interval. As a result, interval-tree then inverted file structure has one interval tree and m (number of unique time intervals in the system) inverted files. Similar to inverted file then interval-tree, we get a set of page lists whose entry is a pair of a time-interval and a textual keyword (this time, time-interval first and then textual keyword).

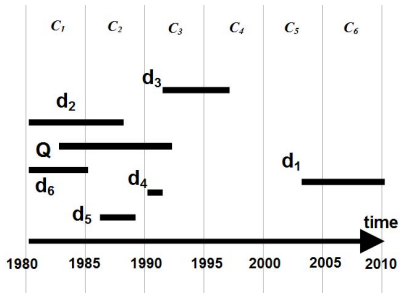
For processing query q, the temporal part of the query T_q is first fed into the interval-tree to find overlapping time-intervals with T_q. For each time-interval, temporal relevance between that time-interval and query time-interval (T_q) is computed and saved. Next, inverted file corresponding to the time-interval (leaf node) is accessed and processed for the textual part of the query K_q. While accessing the lists, the textual score is also calculated. Finally, all lists from all TIKs (time-interval keyword pairs) are merged and textual and temporal scores are combined.

5. SEAMLESS TEMPO-TEXTUAL RANKING

In this section, we define a new scoring mechanism to calculate the temporal relevance and tempo-textual relevance scores. Following the same intuitions and concepts used in regular (textual) searches, we define new concepts and parameters for temporal data. Most notably, inspired by tf-idf in textual context, we define a new scoring mechanism called *temporal tf-idf* for the temporal context. Using (textual) tf-idf scores and temporal tf-idf scores, the tempo-textual relevance is defined and can be used to rank the documents based on both the temporal and textual aspects of the data, simultaneously and efficiently. We discuss two different approaches to calculate the tempo-textual relevance using the temporal tf-idf score. Several variants of the final similarity measure are also presented.

5.1 Temporal tf-idf

In order to be able to use the analogous ideas used in the regular tf-idf score, we need to treat temporal data similar to textual data. Most importantly, we need to represent time which is coherent and continuous in nature, as disjunct and set-oriented units of data - similar to the textual keywords. Hence, we partition the time domain into consecutive cells and assign unique identifiers to each cell. Therefore, each timespan in the document can be associated with a set of cell identifiers. Since we are using overlap as our main temporal query type, these cells are



Example 1 with temporal cells

defined as the cells which overlap with the document timespan. With temporal tf-idf, the overlap of a cell with the document is analogous to the existence of a keyword in the document with tf-idf. However, knowing the overlapping cells is not enough. We need to know how well a cell describes the temporal content of the document. We use the overlap area between each cell and the document to provide a measure of how well that cell describes the document. Analogous to *frequency of term t in document d*, we define *frequency of cell c in document d* as follows: $f_{d,c} = \frac{T_d \cap c}{c}$ which is the area of overlap between the document timespan T_d and cell c divided by the area of cell c. Similar to the frequency of a keyword which describes how well the keyword describes the documents textual contents (K_d), the frequency of a cell describes how well the cell describes the documents temporal contents (T_d). The more the overlap, the better this cell describes the document timespan and viceversa.
Now we can define the following parameters analogous to those of Section 2.2:

(1) $f_{d,c}$: the frequency of cell c in document d

(2) $\max(f_{d,c})$: maximum value of $f_{d,c}$ over all the cells in document d

(3) $\overline{f_{d,c}}$: normalized $f_{d,c}$, which is $\frac{f_{d,c}}{\max(f_{d,c})}$

(4) f_c: the number of documents containing one or more occurrences of cell c

Using the above parameters, we revisit three monotonicity properties discussed in Section 2.2, this time in temporal context: (1) less weight is given to cells that appear in many documents; (2) more weight is given to cells that overlap largely with a document; and (3) less weight is given to documents that contain many cells.
The first property is quantified by measuring the inverse of frequency of a cell c among the documents in the collection. We call this *temporal inverse document frequency* or idf_{temp} score. The second property is quantified by the frequency of cell c in document d (as defined earlier). This is called *temporal term frequency* or tf_{temp} score and describes how well that cell describes the document temporal contents (i.e. T_d). The third property is quantified by measuring the total number of cells in the document and is called *document temporal length*.

Among the above properties, properties (2) and (3) are more intuitive. Property (2) states that more weight should be given to the cells having a large overlap area with the document. The larger the overlap, the better that cell describes the document timespan. For example, in Figure 2, cell c_6 better describes the document d_1 than cell c_5. Property (3) states that less weight should be given to those documents whose timespan covers more cells. Assuming all the other parameters are equal, a document with a smaller coverage (fewer number of cells) should get a higher weight than a document with a larger coverage. Assume two documents one containing history of the world from year 1503 to 2000 and the other one containing history of the world only in 1938. When searching for/about year 1938, the second document should be assigned more weight since it is a better representative of year 1938 than the first document. This is analogous to the fact that in textual context, more weight is given to the documents that contain fewer keywords.

Contrary to properties (2) and (3), property (1) is not very intuitive. It states that less weight is given to the cells appearing in more documents. In the textual context, the *idf* score is a weighting factor determining the importance of each keyword independent of the query. It assigns more weight to keywords appearing in fewer documents, since those are more *meaningful* keywords. However, the definition of *meaningful cell* is not very clear in the temporal context. A popular cell (time) - a cell overlapping with many documents - is a very meaningful cell for some users/applications, while for some others, a distinctive cell (time) - cell appearing in few documents - is more meaningful. To cover both cases, we define two variants of temporal *idf* of cell c: inverse of frequency of a cell c among the documents (*inverted* idf_{temp}) and direct frequency of a cell c among the documents (*direct* idf_{temp}).

5.2 Tempo-Textual Relevance

In this section, we introduce two novel approaches for calculating tempo-textual relevance between a document d and a query q. With the *uni-score* approach, one similarity measure and one document length are used to combine the temporal relevance and textual relevance into one equation. With the *dual-score* approach, temporal and textual relevance are calculated separately, using two document lengths, one for each relevance. Thus a new temporal similarity measure analogous to the textual similarity measure is defined. Both approaches can use the parameter α to assign relative weights.

5.2.1 *Uni-Score Approach.* After partitioning each document timespan to a set of cells, defining the temporal tf-idf score and creating one document temporal length for each document timespan, the cells are ready to be treated in a similar manner as the keywords. We define *term* as the smallest unit of data describing each document which is either a keyword or a cell. If we represent keywords associated with a document d by K_d and the cells associated with the same document by C_d, then the set of terms associated with document d is represented by U_d and defined as follows: $U_d = K_d \cup C_d$.

Simply stated, the document's terms are the union of the document's keywords and cells. For instance, in Example 1: $U_{d_1} = \{\texttt{Iraq}, \texttt{war}, \texttt{c}_5, \texttt{c}_6\}$ (see Figures 1b and 2). In order to be able to define a single similarity measure capturing both the textual and temporal relevance, we define the following parameters:

(1) $f_{d,u}$: the frequency of term u in document $d = \begin{cases} f_{d,k} & \text{if } u \text{ is keyword} \\ f_{d,c} & \text{if } u \text{ is cell} \end{cases}$

(2) $\overline{f_{d,u}}$: the normalized frequency of term u in document $d = \begin{cases} \overline{f_{d,k}} & \text{if } u \text{ is keyword} \\ \overline{f_{d,c}} & \text{if } u \text{ is cell} \end{cases}$

(3) f_u: the number of documents containing occurrences of term $u = \begin{cases} f_k & \text{if } u \text{ is keyword} \\ f_c & \text{if } u \text{ is cell} \end{cases}$

where each parameter gets its value from the corresponding parameter in the time or text domain (based on the term type). For instance, the value of $f_{d,u}$ is equal to $f_{d,k}$ when term is keyword and to $f_{d,c}$ when term is cell. Having defined these new parameters, we can now easily redefine Equation 2, this time using terms instead of keywords. This is a new formulation capturing the

keywords (textual relevance) and the cells (temporal relevance) in a unified manner.

$$
\begin{aligned}
w_{q,u} &= \begin{cases} (1-\alpha).\ln(1+\frac{n}{f_u}) & \text{if } u \text{ is keyword} \\ \alpha.w_{q,c} & \text{if } u \text{ is a cell} \end{cases}; \\
w_{d,u} &= \begin{cases} (1-\alpha).\ln(1+\overline{f_{d,u}}) & \text{if } u \text{ is keyword} \\ \alpha.\ln(1+\overline{f_{d,u}}) & \text{if } u \text{ is cell} \end{cases}; \\
\widehat{W_d} &= \sqrt{\sum_u w_{d,u}^2}; \qquad\qquad \widehat{W_q} = \sqrt{\sum_u w_{q,u}^2}; \\
\widehat{S}_{q,d} &= \frac{\sum_u w_{d,u}.w_{q,u}}{\widehat{W_d}.\widehat{W_q}}.
\end{aligned}
\tag{4}
$$

The variable $w_{d,u}$ captures the *tempo-textual term frequency* score (tf_{tt}). The variable $w_{q,u}$ captures the *tempo-textual inverted document frequency* (idf_{tt}). The parameter α is integrated into the weighting scheme to capture weighted relevance of time versus text. $\widehat{W_d}$ represents *tempo-textual document length* and $\widehat{W_q}$ is (tempo-textual) query length. Finally, $\widehat{S}_{q,d}$ is the similarity measure showing how *tempo-textual relevant* document d is to query q.

5.2.2 *Dual-Score Approach.* In the *uni-score* approach, keywords and cells are treated in exactly the same manner. Keywords and cells tf and idf scores are used in one equation and one similarity measure ($\widehat{S}_{q,d}$) using one document length ($\widehat{W_d}$) to calculate the final relevance score. There might be cases when most of the documents in the collection contain very long document timespans but very few keywords (or vice versa). In this case, it is better to calculate the textual and temporal relevance scores separately. Hence, we discuss another approach to calculate the similarity measure between document d and query q in the tempo-textual context. We first calculate the temporal relevance and the textual relevance of document d and query q independently and then use an aggregation function to compute the overall tempo-textual relevance score. Using the temporal tf-idf parameters and the definitions, we calculate the temporal similarity measure between document d and query q analogous to the textual similarity measure as follows:

$$
\begin{aligned}
w_{q,c} &= \begin{cases} \ln(1+\frac{n}{f_c}) & \text{if inverted document frequency} \\ \ln(1+\frac{f_c}{n}) & \text{if direct document frequency} \end{cases}; \\
w_{d,c} &= \ln(1+\overline{f_{d,c}}); \\
W'_d &= \sqrt{\sum_c w_{d,c}^2}; \qquad\qquad W'_q = \sqrt{\sum_c w_{q,c}^2}; \\
S'_{q,d} &= \frac{\sum_c w_{d,c}.w_{q,c}}{W'_d.W'_q}.
\end{aligned}
\tag{5}
$$

where $S'_{q,d}$ is the temporal similarity measure between document d and query q. This value captures the temporal relevance $tmRel_q(d)$ defined in Section 2.1.
After calculating the temporal relevance using the above equation and computing the textual relevance using Equation 2, the aggregation function F can be used to calculate the final tempo-textual relevance. More formally: $ttRel_q(d) = \alpha.S'_{q,d} + (1-\alpha).S_{q,d}$.

5.3 Variants

We conclude this section by summarizing possible variants of the tempo-textual relevance score. We defined two different approaches to calculate the tempo-textual relevance scores. We also introduced two different ways to define the *temporal idf* factor. Combining our two main approaches with the two definitions of the temporal *idf* score yields four different variants for our final similarity measure:

term u	f_u	type	Tempo-Textual Inverted Index for u
`Iraq`	5	1	$\langle 1,1\rangle\langle 2,1\rangle\langle 3,1\rangle\langle 4,1\rangle\langle 5,1\rangle$
`war`	4	1	$\langle 1,0.33\rangle\langle 2,0.9\rangle\langle 3,0.8\rangle\langle 4,0.54\rangle$
c_1	2	0	$\langle 2,1\rangle\langle 6,1\rangle$
c_2	2	0	$\langle 2,0.6\rangle\langle 5,1\rangle$
c_3	2	0	$\langle 3,1\rangle\langle 4,1\rangle$
c_4	1	0	$\langle 3,0.5\rangle$
c_5	1	0	$\langle 1,0.4\rangle$
c_6	1	0	$\langle 1,1\rangle$

Tempo-textual inverted index for Example 1.

(1) Uni-score with Inverted document frequency (UI)
Where $ttRel_q(d) = \widehat{S_{q,d}}$ and $w_{q,c} = \ln(1 + \frac{n}{f_c})$

(2) Uni-score with Direct document frequency (UD)
Where $ttRel_q(d) = \widehat{S_{q,d}}$ and $w_{q,c} = \ln(1 + \frac{f_c}{n})$

(3) Dual-score with Inverted document frequency (DI)
Where $ttRel_q(d) = \alpha.tmRel_q(d) + (1-\alpha).txRel_q(d)$ and $w_{q,c} = \ln(1 + \frac{n}{f_c})$

(4) Dual-score with Direct document frequency (DD)
Where $ttRel_q(d) = \alpha.tmRel_q(d) + (1-\alpha).txRel_q(d)$ and $w_{q,c} = \ln(1 + \frac{f_c}{n})$

6. TEMPO-TEXTUAL INVERTED INDEX

Tempo-textual inverted index ($\mathrm{T^2I^2}$) is an inverted index capable of indexing and searching both the textual and temporal data in a unified, integrated manner using a single data structure. In this section, we first describe the structure of $\mathrm{T^2I^2}$ and the information it stores. Next, we show how tempo-textual query evaluation is performed using $\mathrm{T^2I^2}$. Two algorithms corresponding to our two approaches are presented. Finally, we discuss briefly how $\mathrm{T^2I^2}$ can be extended to more general cases.

6.1 $\mathrm{T^2I^2}$ Structure

Since $\mathrm{T^2I^2}$ is an inverted index, its structure is very similar to the structure of the regular inverted indexes. $\mathrm{T^2I^2}$ consists of two parts: vocabulary and inverted lists. The vocabulary contains all the *terms* in the system which includes all the (textual) keywords and cells (cell identifiers). For each distinct term, three values are stored in the vocabulary: 1) f_u representing the number of the documents containing the term u, 2) a pointer to the corresponding inverted list and 3) the type of term which is used to help calculate the tf and idf scores. The second component of $\mathrm{T^2I^2}$ is a set of inverted lists each corresponding to a term. For the corresponding term u, each list stores the following values: identifiers of the documents containing term u and the normalized frequencies of term u for each document d. The latter is represented by $\overline{f_{d,u}}$. Figure 2 redraws the Example 1 with temporal cells (each cell is 5 years) and Figure 3 shows the complete $\mathrm{T^2I^2}$ for Example 1.

6.2 Query Processing

As discussed in Section 2.1, the tempo-textual query consists of two parts: the query keywords K_q and the query timespan T_q. To process tempo-textual queries, we first need to convert T_q to a set of cells C_q. C_q is the set of cells overlapping with the document timespan T_q . After calculating C_q, we define the set of terms associated with each query by U_q as follows: $U_q = K_q \cup C_q$.

Algorithms 1 and 2 show the algorithms to perform top-k tempo-textual search using $\mathrm{T^2I^2}$ for the uni-score and the dual-score approaches, respectively. With both algorithms, accumulators are used to store the partial similarity scores. The main difference is that Algorithm 1 uses one accumulator A_d while Algorithm 2 uses two accumulators A_d and A'_d. After all the query terms are processed, similarity scores $\widehat{S}_{q,d}$, $S_{q,d}$ and $S'_{q,d}$ are derived by dividing each accumulator

value by the corresponding values of $\widehat{W_d}$, W_d and W'_d, respectively (first one used in the uni-score algorithm while the last two are used in the dual-score algorithm) . Finally, the k largest documents are identified and returned to the user.

In the uni-score approach (Algorithm 1), we assign one accumulator for each document d which is denoted by A_d. Partial similarity scores are stored in these accumulators. Initially, all the accumulators have a value of zero (e.g., similarity of zero). The query terms are processed one at a time and for term u, the accumulator A_d for each document d included in the u's inverted list is increased by the contribution of u to the similarity of d and q. After all query terms are processed, similarity scores $\widehat{S}_{q,d}$ are derived by dividing each accumulator value by the corresponding value $\widehat{W_d}$. Finally, the k largest documents are identified and returned to the user.

In the dual-score approach (Algorithm 2), two accumulators are assigned to each document: A_d and A'_d. The partial textual similarity score is stored in A_d while the partial temporal similarity score is stored in A'_d. Initially, both accumulators are empty (zero score). Again, terms are processed one at a time and for each term u and for each document d included in u's inverted list, values of A_d and A'_d are increased by the contribution of term u to the textual and temporal similarity of document d to q, respectively. After processing all the query terms, temporal similarity scores $S'_{q,d}$ are calculated by dividing each A'_d to its corresponding W'_d. In addition, textual similarity scores $S_{q,d}$ are computed by dividing the values of A_d accumulators to the corresponding W_d values. Finally, for each document, if both similarity scores are larger than zero, the final similarity score is calculated as the weighted sum of these two scores.[2].

Algorithm 1 top-k tempo-textual search, uni-score

```
Allocate an accumulator A_d for each document d
Set A_d ← 0
for each query term u in q do
   Calculate w_{q,u} and fetch the inverted list for u
   for each pair < d, f_{d,u} > in the inverted list do
      Calculate w_{d,u}
      Set A_d ← A_d + w_{q,u} × w_{d,u}
Read the array of Ŵ_d values
for each A_d > 0 do
   Set Ŝ_d ← A_d ÷ Ŵ_d
Identify the k greatest Ŝ_d values
```

7. GENERALIZATION

In this section, we briefly show how T^2I^2 can be extended into more general cases.

7.1 Multiple Timespans

is that there is no limit on the number of timespans in a document. Instead of treating the document timespan as one long (and maybe sparse) interval, we can use several separate, disjoint time intervals using T^2I^2. This is feasible because our final temporal relevance score can be computed by separately computing the temporal score of each cell intersecting with the various document timespans. Another advantage of T^2I^2 is its capability to represent the document timespan in any arbitrary granularity and not necessarily in common time units (e.g. days, months, years). The only information we need to calculate for temporal tf-idf score is the area of overlap between each cell and each document timespan. The cost is negligible since the computation is happening one-time during the index construction.

[2]In order to perform this algorithm efficiently, we use the *Threshold Algorithm* described in [Fagin *et al.* 2003]

Algorithm 2 top-k tempo-textual search, dual-score

Allocate two accumulators A_d and A'_d for each document d
Set $A_d \leftarrow 0$
Set $A'_d \leftarrow 0$
for each query term u in q **do**
  Calculate $w_{q,u}$ and fetch the inverted list for u
  for each pair $< d, f_{d,u} >$ in the inverted list **do**
    Calculate $w_{d,u}$
    if type of u is a keyword **then**
      Set $A_d \leftarrow A_d + w_{q,u} \times w_{d,u}$
    else
      Set $A'_d \leftarrow A'_d + w_{q,u} \times w_{d,u}$
Read the array of W_d values
for each $A_d > 0$ **do**
  Set $S_d \leftarrow A_d \div W_d$
Read the array of W'_d values
for each $A'_d > 0$ **do**
  Set $S'_d \leftarrow A'_d \div W'_d$
  if $A_d > 0$ **then**
    $\widehat{S_d} = \alpha.S'_d + (1 - \alpha).S_d$
Identify the k greatest $\widehat{S_d}$ values and returns the corresponding documents

7.2 Points

We assumed that each document timespan is an interval. In the context of the web, this is a reasonable assumption, still in the cases when the document temporal feature is only a point in time (p) we can generalize our approach as follows. We find the temporal cell that intersects with p and call it c_p. The new document timespan is c_p plus m temporal cells before and m cells after c_p. The value of m is determined by the user and is usually a small number. In case of multiple points, we can either apply the above algorithm for each point and generate multiple timespans, or use one (possibly long) timespan covering all the points.

7.3 Freshness

We assumed that when a user issues a query using a time-interval, all temporal cells inside the time-interval have the same importance to the user. This assumption is true for most of the historical queries (which is the focus of our paper). However, for some types of queries (e.g., queries in which *end time* is now), it makes more sense to use *time decay* to reduce the importance of the older cells. There exists several studies on *time decay* in the literature [Graham *et al.* 2009]. We can integrate one of the existing approaches to our proposed technique as follows. For each cell, we define a new parameter called *temporal decay factor* or df_{temp}. The weight given to this parameter represents cell's decay and is inverse-polynomial to the cell's elapsed time. The factor df_{temp} is defined as a monotone non-increasing function and as follows: $df_{temp}(c) = (t_c - t_{base} + 1)^{-(t_c - t_{base})}$ where c is the cell id, t_c is the time associated with cell c (e.g., cell's start time) and t_{base} is the time associated with the reference point. Finally, we integrate the inverse of df_{temp} into our methods similar to what we did for idf_{temp}. Consequently, we have three parameters tf_{temp}, idf_{temp} and inverse of df_{temp} impacting the temporal ranking.

7.4 Weights

When querying the system, there are two types of weight factors users may want to manipulate 1) setting different weights to the temporal and textual relevances, and 2) setting different weights to different terms in the query. For the first scenario, we have used the parameter α in this

paper. T^2I^2 can also support setting different weights for different terms. There are several existing methods to solve this problem for textual keywords. Since we are treating cells similar to keywords, those methods can also be applied to the temporal cells. As one possible solution, we define *query term weights* $\alpha_{q,k}$ and $\alpha_{q,c}$ as the weight of keyword k in query q and the weight of cell c in query q, respectively. By multiplying $w_{d,k}$ and $w_{d,c}$ values by $\alpha_{q,k}$ and $\alpha_{q,c}$, respectively, query term weights are integrated into the relevance scores. This opens up a wide array of sophisticated query capabilities for the users.

7.5 Leveraging Existing Search Engines

approach is the fact that it can be integrated into the existing search engines easily and seamlessly. Since the structure of T^2I^2 is very similar to the structure of the regular inverted indexes, the same techniques used in regular search engines (built on inverted indexes) can be applied to our time-based search technique. Structure wise, the main difference is in using T^2I^2 instead of the regular inverted indexes. This essentially translates to using a larger vocabulary (combination of cells and keywords instead of only keywords). The average size of inverted lists would not change since that only depends on the total number of documents, which is fixed. This is very promising because the cost of existing search engines is dominated by the cost of traversing the inverted lists and not the size of the vocabulary. The easy integration of our approach into the existing search engines is not only very beneficial for current search engines but also enables us to optimize T^2I^2 using a large body of work that exists in this field. More interestingly, some of the optimization techniques seem to work better on T^2I^2. For instance, *caching* is another technique used in existing search engines. It is easy to see that with T^2I^2, by caching the inverted lists for the cells nearby the current query cell, we can improve the query performance significantly. It is very likely that nearby cells queried together.

8. EXPERIMENTS

In this section we evaluate the efficiency and accuracy of our proposed approaches in two ways. First, we provide a cost model analysis for the proposed approaches. Next, we present results from simulations based on real document sets.

8.1 Cost Model

In this section, we analyze the search (query processing) cost for T^2I^2 and three hybrid approaches. The symbols used in the cost models are presented in Table I.

8.1.1 *Cost of FnT.* For a query with $|K_q|$ number of keywords and a time-interval T_q, the search cost has three parts: 1) retrieval and processing of the interval-tree and m lists corresponding to m time-intervals, 2) calculating temporal relevance for each overlapping time-interval, and 3) access and retrieval of $|K_q|$ keywords and their corresponding $|K_q|$ lists. In other words:

$Time(FnT) = T_{tree}(FnT) + T_{tRel}(FnT) + T_{list}(FnT)$

The cost of the retrieval and processing of an interval-tree with m leaf nodes is: $T_{tree} = O(log\, m)$. The time to read a list whose length is l from disk is: $T_{list} = O(l/B).T_{I/O}$. For all the approaches, we will ignore the cost of retrieving $|K_q|$ keywords from the vocabulary (we can assume that the vocabulary reside in memory and/or implemented with a simple hash table). Thus:

$$\begin{aligned} Time(FnT) &= T_{tree}(FnT) + T_{tRel}(FnT) + T_{list}(FnT) \\ &= O(log\, m) + \sum_{i=1}^{m'} O(P_T(t_i)/B).T_{I/O} \\ &+ \sum_{i=1}^{m'} T_{temp} + \sum_{i=1}^{|K_q|} O(P_K(k_i)/B).T_{I/O} \end{aligned} \tag{6}$$

Table I: Symbols

Symbol	Meaning
n	total number of documents
m	total number of time-intervals (temporal expressions)
$H(q)$	total number of time-interval-keywords (TIKs) in the query q
$P_K(k_i)$	length of the page list for the keyword k_i
$P_C(c_i)$	length of the page list for the temporal cell c_i
$P_T(t_i)$	length of the page list for the time-interval t_i
$P_H(h_i)$	length of the page list for the TIK h_i
T_{tree}	time cost associated with retrieving an interval-tree
T_{tRel}	time cost associated with calculating temporal relevance
T_{list}	time cost associated with retrieving page lists
T_{temp}	time cost of calculating (one) temporal relevance
$T_{I/O}$	time cost of one disk access
B	page size

where m' is the number of overlapping time-intervals with the query's time-interval and $|K_q|$ is the number of textual keywords in the query.

8.1.2 *Cost of FtT.* Given a query q with temporal and textual parts (T_q and $|K_q|$ respectively) and assuming $H(q)$ is the number of temporal-interval-keywords (TIKs) for query q, the search cost for FtT approach has three parts: 1) retrieval and processing of $|K_q|$ interval-trees, 2) calculating temporal relevance for all leaf nodes of all trees, and 3) access and retrieval of $H(q)$ keywords and $H(q)$ lists corresponding to them (one list per TIK). Assuming $\overline{m}$ is the average number of leaf nodes for interval-trees, we have:

$$\begin{aligned} Time(FtT) &= T_{tree}(FtT) + T_{tRel}(FtT) + T_{list}(FtT) \\ &= |K_q|.O(log\,\overline{m}) + \sum_{i=1}^{H}(q)T_{temp} + \sum_{i=1}^{H(q)} O(P_H(h_i)/B).T_{I/O} \end{aligned} \tag{7}$$

8.1.3 *Cost of TtF.* The search cost of TtF approach is dominated by three parts: 1) retrieval and processing of one interval-tree with m leaf nodes, 2) calculating temporal relevance for overlapping leaf nodes (time-intervals), and 3) access and retrieval of $H(q)$ keywords and $H(q)$ lists corresponding to them.

So:

$$\begin{aligned} Time(TtF) &= T_{tree}(TtF) + T_{tRel}(TtF) + T_{list}(TtF) \\ &= O(log\,m) + \sum_{i=1}^{m'} T_{temp} + \sum_{i=1}^{H(q)} O(P_H(h_i)/B).T_{I/O} \end{aligned} \tag{8}$$

8.1.4 *Cost of T^2I^2.* For T^2I^2, the main search cost is dominated only by one part: 1) access and retrieval of $|K_q| + |T_q|$ terms and $|K_q| + |T_q|$ lists corresponding to them. Note that for all four approaches, textual relevance calculation is done seamlessly during the query processing. For T^2I^2, the temporal relevance computation is also integrated into the search process, similar to the textual relevance calculation. Since, values of $\overline{f_{d,c}}$ (capturing the temporal relevance between cell c and document d) are already calculated and stored in T^2I^2, we do not need to compute the temporal relevance for each document during the search process and on-the-fly. That by itself reduce the search cost for T^2I^2 in comparison with other approaches. Also, since there is no extra tree structure, no cost is associated with retrieval of one or more (interval) trees and access and processing of the different nodes in each tree. We can show the search cost of T^2I^2 as follows:

$$
\begin{aligned}
Time(T^2I^2) &= T_{list}(T^2I^2) \\
&= \sum_{i=1}^{|T_q|} O(P_C(c_i)/B).T_{I/O} \\
&+ \sum_{i=1}^{|K_q|} O(P_K(k_i)/B).T_{I/O}
\end{aligned} \tag{9}
$$

8.2 Performance Experiments

In this section, we present the experimental evaluation of our approaches on real-world datasets in order to study the efficiency of our proposed index structures.

Setting and Dataset: Our experiments use two datasets with their properties summarized in Table II. FREEBASE dataset is generated from data on *freebase* web-site (www.freebase.com). *Freebase* is an online collection of structured data harvested from many sources, including individual wiki contribution. We used *events*[3] data on Freebase. Based on the events' schema definition, "An event is a topic that can be described by the time or date at which it happened. Long-lasting events may be described as occurring between two dates". Among properties of each event (web-page) on *freebase* are attributes *start date* and *end date*, which are used in our experiments as t_b and t_e of each document, respectively.

NY-Times dataset is from *New York Times Annotated Corpus* [4] that contains around 1.8 million articles published in NY-times newspaper between 1987 and 2007. This is the de-facto document set used in most of recent studies in this field. For temporal expressions in the documents, we used the data generated by [Berberich *et al.* 2010]. This is how they extracted temporal information from the content of the documents. Temporal expressions were extracted using TARSQI [Verhagen *et al.* 2005]. TARSQI detects and resolves temporal expressions using a combination of hand-crafted rules and machine learning. It annotates a given input document using the TimeML [tim] markup language. Building on TARSQIs output, they extracted range temporal expressions such as from 1999 until 2002, which TARSQI could not support [Berberich *et al.* 2010]. While the publication time of this dataset is from 1987 to 2007, temporal expression extracted from content of the documents contain time-intervals for a much larger time period (temporal expressions basically could relate to any event in past and future) . For our NY-TIMES dataset, we filtered in documents with time-intervals between 1512 and 2011.

All the index structures are disk resident and the page size is set at 4 KB. For T^2I^2, we partition the time domain to 1-day cells (i.e., each temporal cell is a day). It is interesting to see that cell sizes larger than one day have a much better performance than the default cell size (i.e., one day). All of our experiments are conducted on a machine with an Intel Core2 Duo 3.16 GHz CPU and with 4GB main memory.

8.2.1 *NY-TIMES Dataset.* In this section, we evaluate the performance of T^2I^2 in terms of number of disk IOs and search time for the NY-TIMES dataset. We also perform the same evaluation study and show the results for the IIO, FnT, FtT and TtF approaches.[5] For each query, we randomly choose 1 to 6 keywords from the list of top-1000 most frequent keywords in the dataset, one random *start time* between January 1, 1512 and December 31, 2011 and one random *time-interval length*, which can be one day, one week, one month or one year. Given *start time* and time-interval length of each query, an *end time* is calculated and assigned to the query (i.e., *end time* = *start date* + *timespan length*). Queries are performed in *rounds*. Each round

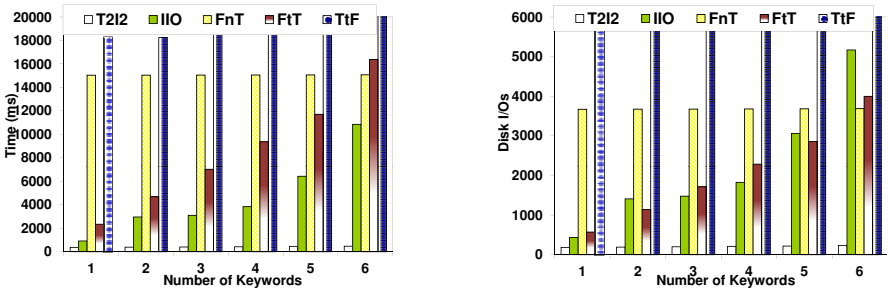
[3] `http://www.freebase.com/type/schema/time/event`

[4] `http://www.ldc.upenn.edu/Catalog/docs/LDC2008T19/`

[5] To have a fair comparison, none of the optimization techniques (early-termination,etc.) are implemented for any of the approaches.

Table II: Dataset Details

Dataset	FREEBASE	NY-TIMES
Total # of documents	34,641	1,855,655
Total # of keywords	2,093,764	423,704,062
Total # of unique keywords	111,038	6,234,465
Average # of unique keywords per document	60	228
Total # of unique time-intervals (temporal expressions)	14,051	128,905
Average # of unique time-intervals (temporal expressions) per document	0.94	3.35
Time range	211 years (1800-2010)	500 years (1512-2011)



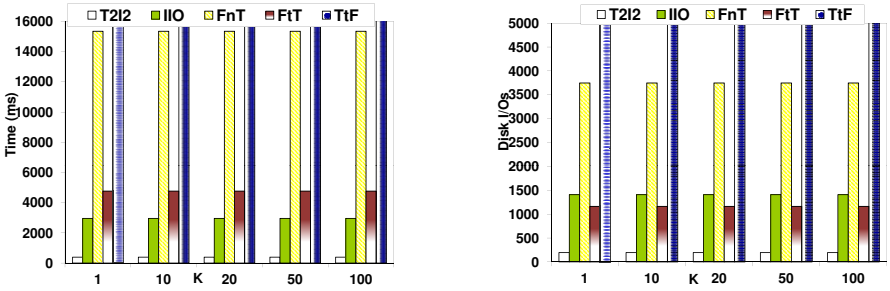
(a) Search Time (b) Disk I/O

Impact of number of keywords on query cost - NY-TIMES

consists of 100 queries and is conducted for each input setting. Value of α is 0.5 unless another value is specified.

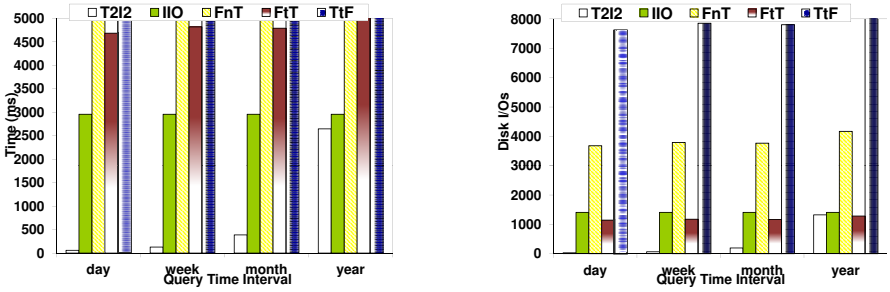
Effect of number of keywords: With the first set of experiments, we evaluate the impact of the number of keywords in each query ($|K_q|$) on the query cost. In this set of experiments, we vary $|K_q|$ from 1 to 6 while fixing k at 10 and query time-interval length at 30 days (one month). For each method, we report the average query cost in processing each round. Figures 4(a) and 4(b) show the results for search time and number of page accesses, respectively. The major observation is that T^2I^2 is significantly superior to all other approaches for all the cases. The other observation is that all five approaches perform worse as the number of keywords increase (as expected). As expected, the impact of the increase in the number of keywords is very significant for FtT and IIO. With FtT, the number of disk IOs increases by a factor of 7 when the number of keywords changes from 1 to 6 and for IIO this factor is around 11. This is because more number of query keywords for FtT results in accessing and traversing more (sometimes very large) trees. For IIO, more number of keywords results in more lists to retrieve and process. More importantly, larger number of candidate documents will be returned from the textual filtering step and as a result more documents need to be retrieved and processed in the second phase.

Effect of k: In this set of experiments, we evaluate the performance of T^2I^2, IIO, FnT FtT and TtF by varying the number of requested results k. We report the average query cost for each round. Here, we fix the number of keywords at 2 and timespan length at 30 days. The value of k varies from 1 to 100. The results are shown in Figures 5(a) and 5(b). The first observation is that T^2I^2 again (significantly) outperforms all other approaches. The second observation is that for all five approaches the query cost does not change much when k increases. This happens because our implementation (for none of the approaches) uses any of the early-termination techniques (as we said earlier, so that we can have a fair comparison). In both figures (similar to Figures



(a) Search Time (b) Disk I/O

Impact of k on query cost - NY-TIMES



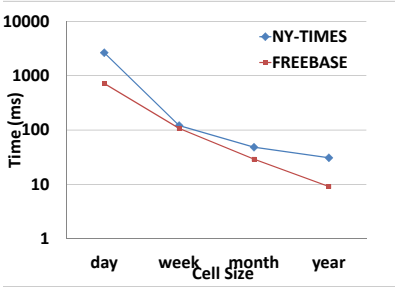
(a) Search Time (b) Disk I/O

Impact of time-interval size on query cost - NY-TIMES

4(a) and 4(b)) TtF performs the worst. TtF performing the worst in almost all the experiments indicates the sensitivity of TtF to the number of size of time intervals in the system (i.e., temporal distribution of data). For both datasets, the number of time intervals in the dataset is fairly large resulting in many leaf nodes and inverted files in TtF. Also, for leaf nodes that represent large time intervals (which is common in our datasets), not much filtering is done based on the temporal aspect of the data and hence inverted files for those leaf nodes will be fairly large.

Effect of time-interval length: In the third set of our experiments, we evaluate the impact of changing the length of the query's time-interval. For different rounds, we set the query's time-interval length to one day, one week (7 days), one month (30 days) and one year (365 days). In this set of experiments, we fix the number of keywords at 2 and k at 10. The results for the search time and the number of disk IOs are shown in Figures 6a and 6b, respectively. Again, for most cases T^2I^2 outperforms the other four approaches. As expected, the query cost increases for T^2I^2, FnT, FtT and TtF as the time-interval length increases. For T^2I^2, increase in the query time-interval size translates into the increase in the number of query terms (temporal cells) and consequently more disk IOs (and search time).

Hence, for the very long query time-intervals with small query cell sizes (e.g., one day) our proposed solution does not *significantly* outperform all other approaches. Even for very long time-intervals, T^2I^2 still significantly outperforms both FnT and TtF because both FnT and TtF need to 1) process huge time-intervals, and 2) retrieve and access large number of leaf nodes and their corresponding (page) lists (due to the size of query interval). FtT needs to traverse smaller trees and smaller time-intervals and IIO results change slightly for different query time-interval lengths, since its performance is not dependant on the temporal part of the query. As we show in Section 8.2.3, increasing the temporal sizes from one day to one week, one month, etc. significantly improve the performance of T^2I^2.



Impact of cell size on query cost

8.2.2 *FREEBASE Dataset.* We also evaluated the performance of T^2I^2 in terms of number of disk IOs and search time for the FREEBASE dataset. Except the scale of the result the trend is very similar to the results reported and discussed for NY-TIMES dataset. Due to the space limitation and similarity of results between NY-TIMES dataset and FREEBASE dataset, we do not report the results from the FREEBASE dataset here.

8.2.3 *Cell Size (Temporal Granularity).* Finally, we show how choosing different values for temporal cell size (temporal granularity) affects the performance of the system. As we noted earlier, we chose one day as the default cell size while constructing T^2I^2 to comply with most of the existing studies and also because day was the smallest unit of time extracted from New York Times Annotated Corpus in [Berberich *et al.* 2010]. To perform this set of experiments, we built T^2I^2 four different time (one for each cell size) and for both datasets (eight time in total). We randomly generated a set of 100 queries for each dataset while fixing k at 10, number of keywords at 2 and query temporal size at 365 for each query. The four cell sizes are: 1 (day), 7 (week), 30 (month) and 365 (year). The results are shown in Figure 7. The main observation for both datasets is that the performance improves as cell size increases. This improvement is the most significant when cell size increases from one day to one week (note that the figures are in logarithmic scale). Larger cell size results in a fewer number of temporal cells and hence fewer retrieval of lists for the same query. For instance, for a query with a time-interval equal to 365 days, when each cell size is one day, we need to access 365 temporal cells and possibly retrieve all their corresponding inverted lists. This number will decrease significantly for larger cell sizes (e.g. 30). One can argue that by increasing cell sizes, the number of documents overlapping with each temporal cell and consequently size of page lists will increase and this will affect the performance. This is true but as it is seen in Figure 7, the impact of the increase in lists is not significant and is dominated by the impact of decrease in number of cells, resulting in overall decrease in the performance cost. There are two reasons for this behaviors. First, for many documents, their time-intervals are large and contain many consecutive temporal cells. As a result, there will be many repeated documents in the lists corresponding to these consecutive cells. When these consecutive cells merge and become one cell (e.g., cell size changes from 1 to 7 days), not many new document will be added to the new cell. Second, increase in the number of documents for each temporal cell, should be significantly large in order to have a significant impact on the number of disk page IOs. This is true because each (disk) page can store a large number of document postings for each inverted list.

8.3 Accuracy Experiments

In this section, we evaluate the effectiveness of our proposed approaches in terms of accuracy (effectiveness).

8.3.1 *Setting and Queries.* **Data.** We used the real-world FREEBASE dataset. As we explained earlier, this dataset contains information regarding events on the freebase website. Orig-

Table III: Queries

	Sports	Politics	Misc.
Short	ncaa men basketball [March 24 2010-April 07 2010]	poland [March 24 1943-April 07 1943]	film festival [September 03 2000-September 15 2000]
	swimming olympics [August 10 2008-August 18 2008]	protests [June 01 2009-June 29 2009]	rock concert [May 15 2009-May 30 2009]
Medium	roger federer [June 01 2008-October 31 2009]	senate election [March 01 2008-August 30 2008]	earthquake [January 15 1990-July 31 1991]
	nfl ravens [August 05 2004-June 31 2005]	bombing [May 15 2007-September 31 2007]	vietnam operation [March 01 1966-April 10 1967]
Long	lakers celtics [August 01 1984-July 29 1986]	german battle [September 03 1916-September 15 1922]	cholera [January 01 1820-December 30 1840]
	fifa world cup [January 01 1958-December 31 1970]	Iraq war [August 01 1980-September 29 1990]	STS shuttle [August 01 2006-June 30 2010]

inally, this data on freebase contained 74,591 events (web-pages). In the processing of this data, we removed the events with no start or end dates, and also removed any event occurred before the year 1800. After these steps, final dataset's size reduced to 34,641 documents.

Queries. For queries, we generated a set of 100 queries from different freebase topics and assigned them timestamps with different granularities in length. The topics were *sports*, *politics* and *misc.*. Timestamp length granularities were ranging from one day to few decades. We categorized the timestamp granularities into three groups: *short* ranging from one day to few weeks, *medium* ranging from one month to few months, and *long* ranging from one year to several years. we had to filter out and/or tune some query timestamps. Finally, we categorized all 100 queries into nine different groups with regards to their timestamp granularity and topic, and randomly selected two queries for each group from our set of 100 queries. All the selected queries are shown in Table III.

Approaches. We computed top-5 query results for each query using four approaches in Section 5: *DI*, *DD*, *UD*, *UI*, and also baseline and hybrid approaches in Section 3: *BH*.[6]

Relevance Assessment. After computing top-5 results for each of our 18 queries using all 5 approaches, we ran a user study using Amazon Mechanical Turk (`https://www.mturk.com/`). One task (hit) was generated for each unique result (web-page). The web-page alongside the query keywords and the query timestamp in regular date format (e.g. November 7th, 1980 to December 12, 1981) were provided to the workers. Workers could choose whether the web-page is *relevant* or *non-relevant*. They could also choose the '*I cannot assess this document*' option (in case their knowledge was not enough to evaluate the document). Workers could also add their comments/explaination for each assessment. Each task (web-page assessment) was assessed by five workers. Each worker was rewarded $0.02 by completion of each assessment. Overall, workers chose *relevant* for 64% of the assessments, *non-relevant* for 33% of the assessments and *I cannot assess this document* for 3% of the assessments.

8.3.2 *Results.* We evaluated the accuracy of the methods under comparison using two standard metrics: Precision at k and nDCG at k. In calculating precision at k, we consider a document *relevant* if the majority of workers assessed that document as relevant and *non-relevant* otherwise. When computing nDCG at k, we consider the average relevance given by the users to each document, interpreting *relevant* as grade 1 and *non-relevant* as 0, respectively.

Overall. The overall result of our relevance assessments with $k = 5$ and using the five approaches under comparison is shown in Table IV. For the precision@5, The first observation is that all of the evaluated methods generate accurate results (precision larger than 0.6) while three of the (seamless) tempo-textual ranking methods (*DI*, *DD* and *UD*) generate *very* accurate results (precision larger than 0.8). The second observation is that, as expected, the dual-score

[6]Note that all three hybrid approaches as well as the baseline approach generate the same final ranking.

Table IV: Precision@k and nDCG@k of various rankings

Method	Precision@5	nDCG@5
DI	0.83	0.74
UI	0.68	0.62
DD	0.88	0.77
UD	0.82	0.74
BH	0.63	0.58

Method	Sports		Politics		Misc.	
	P@5	N@5	P@5	N@5	P@5	N@5
DI	0.73	0.71	0.9	0.75	0.86	0.76
UI	0.53	0.53	0.66	0.59	0.86	0.73
DD	0.8	0.73	0.96	0.77	0.9	0.82
UD	0.76	0.70	0.8	0.69	0.9	0.85
BH	0.53	0.53	0.66	0.58	0.7	0.62

Table V: Precision@k and nDCG@k by topic

Method	Short(days)		Medium(months)		Long(years)	
	P@5	N@5	P@5	N@5	P@5	N@5
DI	0.8	0.76	0.73	0.63	0.96	0.83
UI	0.8	0.70	0.6	0.52	0.66	0.64
DD	0.96	0.82	0.73	0.68	0.96	0.83
UD	1	0.83	0.66	0.67	0.8	0.73
BH	0.76	0.68	0.66	0.57	0.46	0.50

Table VI: Precision@k and nDCG@k by cell size

approaches perform the search more accurately than the uni-score approaches. Using two scores and two document lengths generate more accurate rankings than using one combined score and only one document length. As for the nDCG@5, the above observations are reconfirmed. Three of our tempo-textual ranking methods outperform all other approaches pretty well. Also, baseline and naive (hybrid) approaches gain nDCG@5 of less than 0.7. Again, the dual-score approaches perform better than uni-score approaches.

Topic. For each query topic, we evaluate the effectiveness of each method using the same metrics. In Table V, we show the results of the relevance assessment for each query topic separately. The results are in support of our prior observations. The most accurate method for all three topics is *DD*.

Timestamp Granularity. In this experiment, we present the results of our relevance assessment for different query timespamp lengths. The results are shown in Table VI. As it is clear, there exists significant variations in the accuracy of the approaches across different timestamp granularities. The best ranking varies by timestamp granularity and measure.

9. RELATED WORK

There are several studies on the versioned text data such as web-archives. In *time-travel text search* [Berberich *et al.* 2007a], [Berberich *et al.* 2007b], the goal is to identify and rank relevant documents *as if the collection was in its state as of query time*. In these methods, the final score of each document is usually calculated by aggregating the scores of the document over the query time interval. In other words, the final top-k results are the most textually relevant documents during the query interval (or as of the query time) [Herscovici *et al.* 2007]. Another type of query on the versioned text collections is *durable top-k search*. With this type of search, the goal is to find the documents that are consistently in the top-k throughout the sequence of rankings defined by the query time-interval and the query keywords [Leong Hou *et al.* 2010]. [Norvag *et al.* 2006] focused on a simpler problem of temporal text-containment queries, which is a query for all versions of the documents that contained one or more particular word at a particular time. Temporal text-containment queries ignore the relevance scoring of the results. All of the above approaches work on document collection as a whole and not on specific keyword-temporal queries. Moreover, none of these methods' final scores and rankings are based on both the textual relevance and the temporal relevance. A few other methods use the publication time of documents to improve the relevance ranking. [Li *et al.* 2003] presents a language modeling technique that factors publication time of documents in order to favor recent documents in its relevance ranking. [Corso *et al.* 2005] also takes into account publication time of documents (and also their interlinkage) to rank news articles. On a separate but somehow related topic, [Dai *et*

al. 2010], [Dai] and [Efron 2011] focus on freshness of web documents and study the integration of freshness/recency into the relevance ranking model.

In [Pasca *et al.* 2008], the temporal features of documents are used to help answering time-related questions in open-domain question answering systems. [Dakka *et al.* 2008] proposed a more general framework which detects automatically the important time intervals that are likely to be of interest for time-sensitive queries and leverages documents published within these important time intervals. In [Kalczynski *et al.* 2005], a temporal document retrieval model for business news archives is presented. In [Alonso *et al.* 2006], a new method is presented for clustering and exploring search results based on temporal expressions within the text.

For the past decade, several automatic methods for extracting temporal information from web-documents have been proposed. Using natural language processing (NLP) techniques, specifically information extraction methods, it is possible to identify words or expressions that convey temporal meaning (e.g. today, a long time ago) and use these to date documents [Wong *et al.* 2005]. In [Alonso *et al.* 2007], a three-step approach called *document annotation pipeline* is described. This approach extracts temporal information based on the TimeML standard described in [tim]. TimeML has become the standard markup language for events and temporal expressions in natural language. There are several other tools that can extract temporal information from documents. Examples of such tools are Lingua::EN::Tagger [lin] and TempEX [Mani *et al.* 2000]. For a more detailed study of temporal information extraction techniques, we refer readers to [Verhagen *et al.* 2009] and [Wong *et al.* 2005].

Finally, there are few approaches that consider the temporal information in the documents' content for the relevance ranking and retrieval purposes. In the work of [Baeza-yates *et al.* 2005], the goal is to search for information that points to the future. The presented retrieval model called *future retrieval* uses a simple probability model for future events based on set of time segments and a simple ranking function. In [Jin *et al.* 2008], a temporal search engine (called TISE) supporting content time retrieval for web pages in presented. TISE extracts temporal features from web pages through natural language processing techniques and ranks web-pages using a simple linear combination of their textual relevance, temporal relevance and importance. This is a short paper and does not include the detailed information regarding indexing and query processing steps and how efficient those steps are. In addition, the proposed textual relevance function is basic and does not generate accurate results. In [Jin *et al.*] and [Jin *et al.* 2011], (same) authors propose several hybrid index structures for temporal-textual web search. No ranking function or relevance model is discussed in either paper.

The studies closest to our paper are [Arikan *et al.* 2009] and [Berberich *et al.* 2010]. [Arikan *et al.* 2009] describes how to integrate the temporal expressions into a language modeling approach. Two different approaches (LMF and LMW) are presented to leverage temporal expressions and improve retrieval effectiveness. In a similar paper, [Berberich *et al.* 2010] study how to integrate the temporal expressions into a language model retrieval framework while focussing mostly on the aspect of uncertainty in the meaning of temporal expressions. Both of these studies rank documents according to estimated probability of generating the query, textually and temporally. There are couple of major differences between these two approaches and our proposed study. They both are based on probabilistic language models while our proposal is based on the classical vector space model. As [Baeza-yates *et al.* 2005] noted in its conclusion section, for searching the past a probabilistic model does not make that much sense, as events in the past did (almost always) happen. More importantly, while our proposed solution provides a complete framework for temporal-textual indexing, query processing and relevance ranking, the aforementioned approaches only focus on the relevance ranking part. It is not clear how efficient the indexing and search processes perform for these two studies. Furthermore, as we explained in Section 7, our proposed solution can be easily integrated into existing systems while [Arikan *et al.* 2009] and [Berberich *et al.* 2010] integration mechanism does not seem trivial.

10. CONCLUSIONS

Temporal-textual search is a new field of research concentrating on the integration of time dimension into the textual search engines. In this paper, we introduced the problem of temporal-textual retrieval and proposed a complete framework with both *effective ranking* and *efficient indexing* of temporal and textual features of (web) documents. We proposed a baseline approach and variety of hybrid index structures to exploit popular textual and temporal index structures (i.e., inverted file and interval tree). We proposed a novel index structure called T^2I^2 that handles the temporal and textual features of data efficiently and in a unified manner. Using T^2I^2, we showed how query processing is performed efficiently and also discussed how to extend T^2I^2 into more general cases. We experimentally and analytically evaluated our proposed approaches and showed the high efficiency of T^2I^2.

As part of our future work, we plan to extend T^2I^2 with multi-resolution indexing approaches. As we described in Section 8, having different granularities (resolutions) for the temporal cell sizes of T^2I^2 will further improve the efficiency of our proposed index. Moreover, we plan to combine this work with our previous study on the indexing and ranking of spatial and textual features of web documents [Kho], and propose a new indexing and ranking framework to support searches on space, time and text simultaneously and efficiently.

REFERENCES

Lingua::en::tagger.

Timeml specification language.

Allen, J. F. 1981. An interval-based representation of temporal knowledge. In *IJCAI'81*.

Alonso *et al.*, O. 2006. Clustering of search results using temporal attributes. In *SIGIR*.

Alonso *et al.*, O. 2007. On the value of temporal information in information retrieval. *SIGIR Forum*.

Arikan *et al.*, I. 2009. Time will tell: Leveraging temporal expressions in ir. In *WSDM*.

Baeza-Yates *et al.*, R. 1999. *Modern Information Retrieval*.

Baeza-yates *et al.*, R. A. 2005. Searching the future. *In: SIGIR Workshop MF/IR*.

Berberich *et al.*, K. 2007a. Fluxcapacitor: Efficient time-travel text search. In *VLDB*.

Berberich *et al.*, K. 2007b. A time machine for text search. In *SIGIR*.

Berberich *et al.*, K. 2010. A language modeling approach for temporal information needs. In *ECIR*.

Corso *et al.*, G. M. D. 2005. Ranking a stream of news. *In: WWW*.

Dai *et al.*, N. 2010. Freshness matters: in flowers, food, and web authority. *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*.

Dakka *et al.*, W. 2008. Answering general time sensitive queries. *CIKM*.

Efron, *et al.*, M. 2011. Estimation methods for ranking recent information. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, 495–504.

Fagin *et al.*, R. 2003. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*.

Graham *et al.*, C. 2009. Forward decay: A practical time decay model for streaming systems. *ICDE*.

Herscovici *et al.*, M. 2007. Efficient indexing of versioned document sequences. *AIRS*.

Jin *et al.*, P.

Jin *et al.*, P. 2008. Tise: A temporal search engine for web contents. *Intelligent Information Technology Applications*.

Jin *et al.*, P. 2011. Indexing temporal information for webpages. *Computer Science and Information Systems ComSIS 8,* 3, 711–737.

Kalczynski *et al.*, P. 2005. Temporal document retrieval model for business news archives. *Inf. Process. Manage.*.

Leong Hou *et al.*, U. 2010. Durable top-k search in document archives. In *SIGMOD*.

Li *et al.*, X. 2003. Time-based language models. In *CIKM*.

Mani *et al.*, I. 2000. Robust temporal processing of news. In *ACL*.

Norvag *et al.*, K. 2006. Dyst: Dynamic and scalable temporal text indexing. *ISTRR*.

Pasca *et al.*, M. 2008. Towards temporal web search. In *SAC*.

Preparata *et al.*, F. P. 1985. *Computational Geometry: An Introduction*.

Verhagen *et al.*, M. 2005. Automating temporal annotation with tarsqi. *In: Association for Computational Linguistics*.

Verhagen *et al.*, M. 2009. *Language and Linguistics Compass*.
Wong *et al.*, K. 2005. An overview of temporal information extraction. *Int. J. Comput. Proc. Oriental Lang.*.
Zobel *et al.*, J. 2006. Inverted files for text search engines. *ACM Comput. Surv.*.

Ali Khodaei received his B.S. degree in computer engineering from Iran National University(Shahid Beheshti University), Tehran, Iran, in 2003 and his M.S. degree in information and computer science from University of California, Irvine, in 2006. He has worked as a researcher/developer/manager in several companies and research institutes including Microsoft, Samsung R&D center (SISA), and Integrated Media Systems Center (IMSC). He is currently a Ph.D. candiate in computer science at the University of Southern California. His Research interests include web search and information retrieval, social data analysis, geospatial databases and relevance ranking models.

Cyrus Shahabi is a Professor of Computer Science and Electrical Engineering and the Director of the NSF's Integrated Media Systems Center (IMSC) at the University of Southern California. He was also the CTO and co-founder of a USC spin-off and an InQTel portfolio company, Geosemble Technologies, which was acquired in June 2012. He received his B.S. in Computer Engineering from Sharif University of Technology in 1989 and then his M.S. and Ph.D. Degrees in Computer Science from the University of Southern California in May 1993 and August 1996, respectively. He authored two books and more than two hundred research papers in the areas of databases, GIS and multimedia. He is currently on the editorial board of the VLDB Journal and IEEE Transactions on Knowledge and Data Engineering. Dr. Shahabi is a recipient of the ACM Distinguished Scientist award and the U.S. Presidential Early Career Awards for Scientists and Engineers (PECASE).

Amir Khodaei is an undergraduate student in Electrical Engineering and Computer Sciences at University of California Berkeley. He has recently joined UC Berkeley AMPLab as an undergraduate researcher. His interests include Cloud Computing and Computer Systems.