

Temporal Search in Web Archives

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Klaus Lorenz Berberich
Max-Planck-Institut für Informatik

Saarbrücken
2010

Dekan der
Naturwissenschaftlich-Technischen
Fakultät I

Prof. Dr.-Ing. Holger Herrmanns

Vorsitzender der Prüfungskommission
Berichterstatter
Berichterstatter
Berichterstatter

Prof. Dr. Jens Dittrich
Prof. Dr.-Ing. Gerhard Weikum
Prof. Dr. Bernhard Seeger
Prof. Dr. Michalis Vazirgiannis

Beisitzer
Tag des Promotionskollquiums

Dr.-Ing. Martin Theobald
19.07.2010

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.

Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, den 10.02.2010

(Klaus Lorenz Berberich)

Abstract

Web archives include both archives of contents originally published on the Web (e.g., the Internet Archive) but also archives of contents published long ago that are now accessible on the Web (e.g., the archive of The Times). Thanks to the increased awareness that web-born contents are worth preserving and to improved digitization techniques, web archives have grown in number and size. To unfold their full potential, search techniques are needed that consider their inherent special characteristics.

This work addresses three important problems toward this objective and makes the following contributions:

- We present the Time-Travel Inverted index (TTIX) as an efficient solution to *time-travel text search* in web archives, allowing users to search only the parts of the web archive that existed at a user's time of interest.
- To counter negative effects that *terminology evolution* has on the quality of search results in web archives, we propose a novel query-reformulation technique, so that old but highly relevant documents are retrieved in response to today's queries.
- For *temporal information needs*, for which the user is best satisfied by documents that refer to particular times, we describe a retrieval model that integrates temporal expressions (e.g., "in the 1990s") seamlessly into a language modeling approach.

Experiments for each of the proposed methods show their efficiency and effectiveness, respectively, and demonstrate the viability of our approach to search in web archives.

Kurzfassung

Webarchive bezeichnen einerseits Archive ursprünglich im Web veröffentlichter Inhalte (z. B. das Internet Archive), andererseits Archive, die vor langer Zeit veröffentlichter Inhalte im Web zugreifbar machen (z. B. das Archiv von The Times). Ein gewachsenes Bewusstsein, dass originär digitale Inhalte bewahrenswert sind, sowie verbesserte Digitalisierungsverfahren haben dazu geführt, dass Anzahl und Umfang von Webarchiven zugenommen haben. Um das volle Potenzial von Webarchiven auszuschöpfen, bedarf es durchdachter Suchverfahren.

Diese Arbeit befasst sich mit drei relevanten Teilproblemen und leistet die folgenden Beiträge:

- Vorstellung des Time-Travel Inverted indeX (TTIX) als eine Erweiterung des invertierten Index, um *Zeitreise-Textsuche* auf Webarchiven effizient zu unterstützen.
- Eine neue Methode zur automatischen Umformulierung von Suchanfragen, um negativen Auswirkungen entgegenzuwirken, die eine fortwährende *Terminologieveränderung* auf die Ergebnisgüte beim Suchen in Webarchiven hat.
- Ein Retrieval-Modell, welches speziell auf *Informationsbedürfnisse mit deutlichem Zeitbezug* ausgerichtet ist. Dieses Retrieval-Modell bedient sich in Dokumenten enthaltener Zeitbezüge (z. B. "in the 1990s") und fügt diese nahtlos in einen auf Language Models beruhenden Retrieval-Ansatz ein.

Zahlreiche Experimente zeigen die Effizienz bzw. Effektivität der genannten Beiträge und demonstrieren den praktischen Nutzen der vorgestellten Verfahren.

Summary

Web archives include both archives of contents originally published on the Web (e.g., the Internet Archive ¹) but also archives of contents published potentially a long time ago that are now made accessible on the Web (e.g., the archive of The Times ²). An increased awareness that web-born contents are worth preserving and improved digitization techniques have led to recent growth of web archives both in their number and individual size. The Internet Archive and the archive of The Times, our aforementioned examples, reveal two important characteristics of web archives and challenges when dealing with them. First, web archives are often large in size containing at least millions or even billions of documents – at the time of writing the Internet Archive has archived 150 billion copies of web pages since 1996. Second, they often cover long time spans – the earliest documents in the archive of The Times were published in 1785. Existing search techniques, though, do not consider these special characteristics inherent to web archives. Therefore, to unfold the full potential of web archives and make them truly valuable resources, efficient and effective tools to access and search them are needed. We address three important problems toward this objective:

Efficiently supporting *time-travel text search* is the first problem that we address. Time-travel text search allows users to issue queries, like those issued against commercial web search-engines, but lets users specify an additional time of interest (e.g., a day in the past), and searches only on the portion of the web archive that existed at the specified time. For instance, to identify relevant contemporary documents about the FIFA World Cup 2006, a user could issue the query `fifa world cup@July 2006` to search only those document versions that existed in July 2006. We propose the time-travel inverted index as an efficient solution to time-travel text search. The time-travel inverted index builds on the well-studied inverted index and extends it to record when data was valid. To

¹ <http://www.archive.org>

² <http://archive.timesonline.co.uk>

address the large scale of web archives mentioned above, we propose temporal coalescing techniques that reduce index size substantially by exploiting the typically high degree of redundancy between document versions. Furthermore, we describe techniques that allow fine-tuning the time-travel inverted index with regard to performance requirements or space constraints by partitioning and replicating data along the time axis. All temporal coalescing techniques and partitioning strategies are formulated as optimization problems, and we describe algorithms to their optimal and –if needed– approximate solution. We demonstrate the practical efficiency of the time-travel inverted index through comprehensive experiments on three representative real-world web archives.

Terminology may evolve drastically during the time spanned by a web archive. As a consequence of this *terminology evolution*, there is a widening gap between the terminology that today’s users employ to formulate queries and the terminology of archived documents. When using existing retrieval techniques, old but still highly relevant archived documents are often not found in response to today’s queries. As a concrete example, for the query **saint petersburg museum**, documents from the 1970s that contain valuable information about museums in Leningrad would often not be found. To counteract this deterioration of result quality, we propose a novel query-reformulation method. The method exploits time-dependent term co-occurrence statistics to determine terms prevalent in the past that used to have a meaning similar to terms in the query issued today. This is accomplished by comparing time-dependent contexts of frequently co-occurring terms – **leningrad** and **saint petersburg** both occur frequently together with, for instance, **russia**, **hermitage**, and **tsar**. Using a Hidden Markov Model, the method identifies good query reformulations by assembling identified terms that are coherent (i.e., make sense when put together) and popular (i.e., part of general language use), again leveraging time-dependent collection statistics. We describe an efficient implementation of the proposed method that achieves interactive response times and demonstrate the method’s practical usefulness on a real-world dataset.

As a third problem, which is one that is not only of interest when searching web archives, we address so-called *temporal information needs*. These are information needs with a strong temporal component that are typically best satisfied by documents that refer to particular times. Consider, as a concrete example,

the query `crusades 13th century`. Existing retrieval models often fail for temporal information needs, since they are unaware of temporal expressions (e.g., “in 1202”) contained in documents and their semantics. One challenge when dealing with temporal expressions is their inherent uncertainty – our example temporal expression “in 1202” may refer to a particular day but also to the year as a whole. We propose a novel retrieval model that is aware of temporal expressions with their semantics and inherent uncertainty and integrates them seamlessly into a language modeling approach to information retrieval. Central to our approach is the formal representation of temporal expressions as the sets of exact time intervals that they may refer to, which captures their inherent uncertainty. Building on that, we devise a generative model for these exact time intervals and follow a query-likelihood approach to estimate the relevance of a document to a given query that contains a temporal expression. Experiments on two real-world datasets with queries and relevance assessments obtained from real users by means of a crowdsourcing platform show that our method yields substantial improvements in result quality for temporal information needs.

Zusammenfassung

Webarchive bezeichnen einerseits Archive ursprünglich im Web veröffentlichter Inhalte (z. B. das Internet Archive ¹), andererseits Archive, die vor möglicherweise langer Zeit veröffentlichte Inhalte im Web zugreifbar machen (z. B. das Archiv von The Times ²). Ein gewachsenes Bewusstsein, dass originär digitale Inhalte (z. B. Webseiten) bewahrenswert sind, sowie verbesserte Digitalisierungsverfahren, haben dazu geführt, dass Anzahl und Umfang von Webarchiven in den letzten Jahren zugenommen haben. Die beiden zuvor genannten Beispiele, das Internet Archive und das Archiv von The Times, zeigen zwei wichtige Merkmale von Webarchiven: Erstens sind diese häufig umfangreich und umfassen Millionen oder gar Milliarden von Dokumenten – seit 1996 hat das Internet Archive 150 Milliarden Kopien von Webseiten gespeichert. Zweitens decken Webarchive oft einen sehr langen Zeitraum ab – so stammen die ältesten der im Archiv von The Times enthaltenen Artikel aus dem Jahr 1785. Bestehende Suchverfahren berücksichtigen diese Charakteristika von Webarchiven derzeit nicht. Damit Webarchive ihr volles Potenzial ausschöpfen und zu wirklich wertvollen Wissensquellen werden können, bedarf es durchdachter Suchverfahren. Die vorliegende Arbeit befasst sich mit den drei folgenden relevanten Teilproblemen:

Zeitreise-Textsuche effizient zu unterstützen ist das erste dieser Teilprobleme. Die sogenannte *Zeitreise-Textsuche* erlaubt es Benutzern, Anfragen zu formulieren, wie sie auch an Websuchmaschinen gestellt werden, diese jedoch zusätzlich mit einer Zeit von Interesse (z. B. einem Tag im vergangenen Jahr) zu versehen. Nur jener Teil des Webarchives, der zum genannten Zeitpunkt existiert hat, wird für eine solche *Zeitreise-Suchanfrage* betrachtet. Sucht man beispielsweise nach zeitgenössischen Dokumenten zur FIFA Weltmeisterschaft im Jahr 2006, so kann man die Suchanfrage `fifa world cup@July 2006` formulieren und damit nur

¹ <http://www.archive.org>

² <http://archive.timesonline.co.uk>

auf jenen Dokumenten suchen, die im Juli 2006 existiert haben. Wir beschreiben einen effizienten Ansatz zur Unterstützung von Zeitreise-Suchanfragen. Unser Ansatz basiert auf dem bekannten invertierten Index und erweitert ihn derart, dass Gültigkeitszeiten von Daten erfasst werden. Um Indizes trotz des großen Umfangs von Webarchiven kompakt zu halten, stellen wir Verfahren zur Verschmelzung von zu aufeinanderfolgenden Versionen des selben Dokumentes gehörenden Daten vor. Ein weiterer Beitrag sind Verfahren, welche eine Feinabstimmung des Index im Hinblick auf zu erfüllende Leistungsgarantien oder Speicherbeschränkungen erlauben. Diese Verfahren basieren auf einer zeitlichen Partitionierung und Replizierung der im Index vorhandenen Daten. Sowohl die Verfahren zur zeitlichen Verschmelzung als auch die Partitionierungsverfahren sind als Optimierungsprobleme formalisiert. Wir beschreiben Algorithmen zur Berechnung von optimalen und –wenn sinnvoll– annäherungsweisen Lösungen. Anhand von umfangreichen Experimenten auf drei repräsentativen Webarchiven wird gezeigt, dass der vorgeschlagene Ansatz praktikabel ist.

Während des von einem Webarchiv abgedeckten Zeitraumes kann sich die gängige Terminologie deutlich verändert haben. Diese *Terminologieveränderung* ist ursächlich für eine sich weitende Kluft zwischen heute gängiger Terminologie, die von Benutzern verwendet wird, um Suchanfragen zu formulieren, und jener Terminologie, in der archivierte Dokumente geschrieben wurden. Für die Suchanfrage *saint petersburg museum* beispielsweise werden in den 1970ern veröffentlichte Dokumente, die wertvolle Information über Museen in Leningrad enthalten, oft nicht gefunden. Um einer daraus resultierenden Verminderung der Ergebnisgüte entgegenzuwirken, stellen wir ein Verfahren zur automatischen Umformulierung von Suchanfragen vor. Anhand von zeitabhängigen Statistiken über das gemeinsame Auftreten von Termen, ermittelt unser Verfahren solche Terme, die in der Vergangenheit eine ähnliche Bedeutung hatten wie die in der vorliegenden Anfrage enthaltenen Terme. Hierzu werden zeitabhängige Kontexte von Termen verglichen – der Term *leningrad* beispielsweise trat in der Vergangenheit häufig mit Termen wie *russia*, *hermitage* und *tsar* auf, genau wie es heute der Term *saint petersburg* tut. Unter Verwendung eines Hidden Markov Modells bestimmt das Verfahren dann gute Umformulierungen der vorliegenden Suchanfrage, indem es solche als ähnlich identifizierte Terme zusammensetzt, welche kohärent (d. h. zusammen Sinn ergebend) und populär (d. h. gängig verwendet) sind. Hierzu greift das Verfahren wiederum

auf zeitabhängige Termstatistiken zurück. Wir beschreiben wie sich das vorgeschlagene Verfahren implementieren lässt, so dass interaktive Antwortzeiten erreicht werden, und demonstrieren seinen praktischen Nutzen.

Beim dritten Teilproblem, das nicht nur für die Suche in Webarchiven von Interesse ist, handelt es sich um *Informationsbedürfnisse mit deutlichem Zeitbezug*. Diese lassen sich häufig am besten durch Dokumente bedienen, welche auf eine bestimmte Zeit Bezug nehmen. Ein konkretes Beispiel stellt die Suchanfrage *crusades 13th century* dar. Bestehende Verfahren scheitern oft an solchen Informationsbedürfnissen, da ihnen in Dokumenten enthaltene Zeitbezüge (z. B. "in 1202") und deren Bedeutung verborgen bleiben. Eine Schwierigkeit beim Umgang mit Zeitbezügen ist deren inhärente Unschärfe – so kann der genannte Zeitbezug "in 1202" sowohl auf einen bestimmten Tag, aber auch auf das gesamte Jahr verweisen. Das vorgestellte Retrieval-Modell berücksichtigt Zeitbezüge, deren Bedeutung sowie die ihnen inhärente Unschärfe und fügt diese nahtlos in einen auf Language Models beruhenden Retrieval-Ansatz ein. Die zentrale Idee dieses Modelles liegt darin, Zeitbezüge als Mengen jener exakten Zeitintervalle zu repräsentieren, auf die sie verweisen können, und damit ihre Unschärfe zu erfassen. Hierauf aufbauend entwerfen wir ein generierendes Modell für Zeitintervalle und verfolgen einen Query-Likelihood-Ansatz, um die Relevanz eines Dokumentes zu einer Suchanfrage mit Zeitbezug zu schätzen. Umfangreiche Experimente auf zwei Dokumentensammlungen unter Verwendung von Suchanfragen und Relevanzbewertungen, die wir mittels Online-Befragung echter Benutzer gesammelt haben, zeigen, dass unser Verfahren eine wesentliche Verbesserung der Ergebnisgüte für die betrachteten Informationsbedürfnisse mit deutlichem Zeitbezug erzielt.

Contents

| | | |
|----------|----------------------------------------------------------|-----------|
| 1 | Introduction | 1 |
| 1.1 | Motivation | 1 |
| 1.2 | Contributions | 3 |
| 1.3 | Publications | 4 |
| 1.4 | Outline | 6 |
| 2 | Foundations & Technical Background | 7 |
| 2.1 | Information Retrieval | 7 |
| 2.1.1 | Retrieval Models | 8 |
| 2.1.2 | Link Analysis | 13 |
| 2.1.3 | Indexing & Compression | 14 |
| 2.1.4 | Query Processing | 17 |
| 2.1.5 | Evaluation | 20 |
| 2.2 | Natural Language Processing | 23 |
| 2.2.1 | Semantic Similarity | 24 |
| 2.2.2 | Hidden Markov Models | 26 |
| 2.2.3 | Temporal Information Extraction | 28 |
| 2.3 | Temporal Databases | 32 |
| 2.3.1 | Indexing Techniques | 33 |
| 2.3.2 | Temporal Coalescing | 39 |
| 2.4 | Web Archiving | 40 |
| 3 | Efficient Time-Travel Text Search in Web Archives | 43 |
| 3.1 | Motivation & Problem Statement | 43 |
| 3.2 | Related Work | 46 |
| 3.3 | Model | 48 |
| 3.3.1 | Time Domain & Collection Model | 48 |
| 3.3.2 | Query Model | 49 |
| 3.3.3 | Retrieval Model | 49 |

| | | |
|----------|--------------------------------------------------------------|------------|
| 3.4 | Time-Travel Inverted Index | 50 |
| 3.5 | Query Processing | 53 |
| 3.5.1 | Time-Point Queries | 53 |
| 3.5.2 | Time-Interval Queries | 54 |
| 3.6 | Temporal Coalescing | 58 |
| 3.6.1 | Boolean Payloads | 59 |
| 3.6.2 | Scalar Payloads | 60 |
| 3.6.3 | Positional Payloads | 64 |
| 3.7 | Partitioning Strategies | 73 |
| 3.7.1 | Performance-Optimal Approach | 75 |
| 3.7.2 | Space-Optimal Approach | 76 |
| 3.7.3 | Performance-Guarantee Approach | 76 |
| 3.7.4 | Space-Bound Approach | 83 |
| 3.8 | Management of Time-Dependent Collection Statistics | 87 |
| 3.9 | FLUXCAPACITOR Prototype Implementation | 89 |
| 3.9.1 | Web-Based GUI | 90 |
| 3.9.2 | FLUXCAPACITOR Server | 90 |
| 3.9.3 | Versioned Document Collection Preprocessing | 93 |
| 3.10 | Experimental Evaluation | 93 |
| 3.10.1 | Setup | 94 |
| 3.10.2 | Datasets | 94 |
| 3.10.3 | Index Size | 99 |
| 3.10.4 | Result Accuracy | 106 |
| 3.10.5 | Query-Processing Performance | 112 |
| 3.11 | Discussion & Outlook | 127 |
| 4 | Terminology Evolution in Web Archives | 129 |
| 4.1 | Motivation & Problem Statement | 129 |
| 4.2 | Related Work | 132 |
| 4.3 | Model | 134 |
| 4.3.1 | Time Domain & Collection Model | 134 |
| 4.3.2 | Collection Statistics | 134 |
| 4.4 | Across-Time Semantic Similarity | 135 |
| 4.5 | Query Reformulation | 137 |
| 4.6 | Implementation | 142 |
| 4.7 | Experimental Evaluation | 144 |

| | | |
|----------|----------------------------------------------------------|------------|
| 4.7.1 | Setup & Dataset | 144 |
| 4.7.2 | Across-Time Semantically Similar Terms | 145 |
| 4.7.3 | Query Reformulation Results | 148 |
| 4.8 | Discussion & Outlook | 150 |
| 5 | Retrieval Models for Temporal Information Needs | 153 |
| 5.1 | Motivation & Problem Statement | 153 |
| 5.2 | Related Work | 156 |
| 5.3 | Model | 157 |
| 5.3.1 | Time Domain & Temporal Expression Model | 157 |
| 5.3.2 | Collection & Query Model | 158 |
| 5.4 | Language Models for Temporal Information Needs | 159 |
| 5.4.1 | Uncertainty-Ignorant Language Model | 162 |
| 5.4.2 | Uncertainty-Aware Language Model | 163 |
| 5.5 | Experimental Evaluation | 167 |
| 5.5.1 | Setup & Datasets | 167 |
| 5.5.2 | Experimental Results | 173 |
| 5.6 | Discussion & Outlook | 177 |
| 6 | Conclusions | 179 |
| | Bibliography | 183 |
| | A Query Workloads | 201 |
| | List of Figures | 207 |
| | List of Tables | 209 |
| | List of Algorithms | 211 |
| | Index | 213 |

Chapter 1

Introduction

1.1 Motivation

Since its advent in the 1990s the Web has affected many aspects of our life and society. Preservation of cultural heritage is one of them. On the one hand, thanks to reduced storage costs and improved digitization techniques, archives of contents that were originally published a long time ago in analog formats are now being digitized and made available on the Web. Consider, as an example, the archive of the British newspaper *The Times* [TIMES] that contains more than two centuries' worth of newspaper articles with the earliest published in 1785. On the other hand, there is a growing awareness that born-digital contents, such as web pages, are part of our cultural heritage and therefore worth preserving. One example in this direction is the *Internet Archive* [IA] that, on its mission to preserve the publicly accessible Web, has collected more than 150 billion web pages since its inception in the year 1996.

These are two examples of *web archives*. We assume a broad notion of *web archive* in this work, which includes archives of content originally published on the Web, but also archives of documents that are accessible on the Web. The two examples above reveal two important properties of web archives. First, they are often *very large* in size, containing many millions if not billions of archived documents. Second, they can *cover a long time span* with publication times of comprised documents spanning decades if not centuries. Apart from that, since new content is continuously produced, web archives keep growing both in size and temporal coverage. Although a preservation of documents is a first important step, however, it is clearly not enough. For web archives to become a truly valuable resource, users must be provided with *efficient and effective* tools to search

and access them. This work addresses three important problems toward this objective:

- (I) Commercial search engines like Google, Yahoo!, and Bing have become indispensable guides on the *current* Web, and users have become accustomed to typing in a few keywords and expecting results of high relevance to their respective information needs. Providing equally powerful search functionality on web archives is desirable. However, a naïve adaptation of the techniques underlying these search engines would ignore the temporal dimension inherent to web archives. Instead, we argue that text search in web archives should be augmented with time-travel functionality, so that queries can be evaluated only on those parts of the web archive that existed at a user's time of interest.

Efficient and scalable support for time-travel text search in web archives is the first problem that this work addresses.

- (II) Web archives, as explained above, may span decades if not centuries. During such long time periods, terminology and general language evolve drastically. Consider, for example, the city of Saint Petersburg that was known as Leningrad until 1991. When using standard retrieval techniques, a user issuing the keyword query `saint petersburg museum`, would not be presented old but potentially highly relevant documents that contain details about museums in Leningrad. Terminology evolution thus negatively affects retrieval effectiveness, i.e., the quality of query results. The reason for this is that today's users formulate queries using today's terminology. Documents in our web archive, on the other hand, were written using terminology prevalent in the past. Old but highly relevant documents are therefore often not found in response to queries issued by today's users.

Dealing with terminology evolution and countering its negative effects on retrieval effectiveness is the second problem that this work addresses.

- (III) Users' information needs often have a temporal component. Consider, as two examples, a sports fan interested in the upcoming 2010 FIFA World Cup that will be held in South Africa, or a historian interested in crusades during the 13th century. For these two temporal information needs, it is unlikely that our web archive contains documents published at the user's time of interest, i.e., documents which we could find using time-travel text

search. Still, there could be many relevant documents in our web archive, namely those whose content specifically refers to the user’s time of interest. For instance, for the second information need, a document providing details on the fourth crusade in 1202 would be relevant. Existing retrieval models, though, often fail for such temporal information needs, where the user is satisfied best by document that refer to particular times, since they are not aware of the semantics inherent to temporal expressions like “13th century” and “in 1202”.

Improving retrieval effectiveness for temporal information needs is the third problem that this work addresses.

1.2 Contributions

This work makes the following key contributions toward the solution of the three important problems defined above:

- (I) We present a comprehensive approach to efficient time-travel text search in web archives and other versioned document. In detail:
 - a) The Time-Travel Inverted index (TTIX) as a versatile framework to index and search versioned document collections.
 - b) Temporal coalescing techniques aimed at supporting different types of queries while keeping the index compact.
 - c) Partitioning strategies that allow tuning the index according to imposed performance requirements or space constraints.
 - d) A comprehensive experimental evaluation on (i) the revision history of the English Wikipedia, (ii) a subset of the data collected by the European Archive, and (iii) the New York Times Annotated Corpus as three representative, long-time, and large-scale real-world web archives.
- (II) We propose a novel query reformulation technique that counters the negative effects that terminology evolution has on retrieval effectiveness in web archive search. The efficiency and effectiveness of our approach is demonstrated using the New York Times Annotated Corpus.
- (III) We introduce a novel retrieval model that integrates temporal expressions seamlessly into a language modeling approach to information retrieval.

Experiments on the English Wikipedia and the New York Times Annotated Corpus show that our approach substantially improves retrieval effectiveness for temporal information needs.

1.3 Publications

The results presented in this work have appeared in preliminary form in several publications. In the following, we briefly summarize these publications and point out their connection to subsequent chapters.

Efficient Time-Travel Text Search in Web Archives

In [BBW07b], we address search in versioned document collections such as web archives. We investigate query types and aggregations that make sense on these collections and propose a method to their efficient evaluation.

- [BBW07b]: Klaus Berberich, Srikanta Bedathur, and Gerhard Weikum. *Efficient Time-Travel Text Search on Versioned Text Collections*. BTW, 2007.

In [BBNW07b], our focus is specifically on time-point keyword queries. The time-travel inverted index is proposed as a versatile framework for indexing versioned document collections. We further investigate how the typically high level of redundancy can be leveraged for compression. Apart from that, we propose partitioning strategies that allow tuning the index according to performance requirements or space limits, respectively.

- [BBNW07b]: Klaus Berberich, Srikanta Bedathur, Thomas Neumann, and Gerhard Weikum. *A Time Machine for Text Search*. SIGIR, 2007.

The implementation of our approach in a prototype system coined FLUXCAPACITOR is described in [BBNW07a].

- [BBNW07a]: Klaus Berberich, Srikanta Bedathur, Thomas Neumann, and Gerhard Weikum. *FluxCapacitor: Efficient Time-Travel Text Search*. VLDB (Demo), 2007.

In [BBW07a, BBWV07], we investigate how time-evolving PageRank scores, as one example of time-evolving collection statistics, can be managed in a way that has a small storage footprint but allows for efficient runtime access.

- [BBWV07]: Klaus Berberich, Srikanta Bedathur, Michalis Vazirgiannis, and Gerhard Weikum. *Comparing Apples and Oranges: Normalized PageRank for Evolving Graphs*. WWW (Poster), 2007.
- [BBW07a]: Klaus Berberich, Srikanta Bedathur, and Gerhard Weikum. *A Pocket Guide to Web History*. SPIRE, 2007.

The techniques proposed in [BBW08] are essential to efficiently support phrase queries. Again, the high level of redundancy is leveraged to reduce index size.

- [BBW08]: Klaus Berberich, Srikanta Bedathur, and Gerhard Weikum. *Tunable Word-Level Index Compression for Versioned Corpora*. EIIR, 2008.

Terminology Evolution in Web Archives

In [BBSW09], we discuss the negative effects that terminology evolution induces when searching web archives. We propose an approach to automatically reformulate queries, so that these negative effects are alleviated.

- [BBSW09]: Klaus Berberich, Srikanta Bedathur, Mauro Sozio, and Gerhard Weikum. *Bridging the Terminology Gap in Web Archive Search*. WebDB, 2009.

Retrieval Models for Temporal Information Needs

In [ABB09b], we propose an initial approach that leverages temporal expressions contained in documents to improve retrieval effectiveness for temporal information needs.

- [ABB09b]: Irem Arıkan, Srikanta Bedathur, and Klaus Berberich. *Time Will Tell: Leveraging Temporal Expressions in IR*. WSDM (Late-Breaking Results), 2009.

We further refined the approach and conducted a comprehensive experimental evaluation to demonstrate its practical usefulness in [BBAW10].

- [BBAW10] Klaus Berberich, Srikanta Bedathur, Omar Alonso, and Gerhard Weikum. *A Language Modeling Approach for Temporal Information Needs*. ECIR, 2010.

1.4 Outline

The remainder of this thesis is organized as follows. Chapter 2 establishes important foundations from different areas of computer science and provides technical background on web archives. Chapter 3 presents and evaluates our approach to time-travel text search in web archives and other versioned document collections. Chapter 4 addresses terminology evolution and its negative effects on web archives. Chapter 5 describes our novel retrieval model that targets temporal information needs and integrates temporal expressions seamlessly into a language modeling approach to information retrieval. Finally, in Chapter 6, we conclude this work and point out interesting directions of future research.

Chapter 2

Foundations & Technical Background

The contributions described in subsequent chapters build on techniques from different areas of computer science. In this chapter, we therefore recapitulate fundamental techniques from these areas that we deem essential for an understanding of this work. Furthermore, we provide some technical background on web archiving as the practical application that motivates the problems addressed in this work.

2.1 Information Retrieval

Information retrieval (IR) as an academic field is fairly broad and deals with different types of information (e.g., text documents, images, and videos) and different problems (e.g., searching, organization, and storage). The primary focus of the field has been on searching text documents (e.g., web pages and newspaper articles), i.e., on finding those text documents in a document collection that satisfy a user's information need. Our focus in this work will also be on text documents, which is why we adapt the following definition from Manning et al. [MRS08] as our working definition for the scope of this work:

Information retrieval (IR) is finding text documents that satisfy a user's information need from within a large document collection.

2.1.1 Retrieval Models

Our above definition of information retrieval leaves two important questions open, namely, (i) how the text documents and users' information needs should be *modeled*, and (ii) how we can decide whether a document satisfies an information need, i.e., whether the document is *relevant*. Different retrieval models, proposed over the past decades, come up with different answers to these two fundamental questions. We next discuss some of these retrieval models and their proposed answers.

Boolean Retrieval

Earliest among the retrieval models is the Boolean retrieval model that views documents as sets of terms. Let \mathcal{V} be the *vocabulary* as the set of terms occurring in the document collection. Each document d in our document collection \mathcal{D} is thus a subset of the vocabulary, i.e., $d \subseteq \mathcal{V}$. Queries in the Boolean model are Boolean expressions, i.e., combinations of terms using the Boolean operators \neg , \wedge , and \vee . The Boolean retrieval model considers a document relevant to an information need if the document satisfies the Boolean expression corresponding to the user's query.

The notion of relevance in the Boolean model is thus inherently binary, i.e., a document is considered either relevant or irrelevant to an information need. All documents that satisfy the query are considered equally relevant and, as an effect, there is no ranking of the result documents.

Vector Space Model

If the document collection is small or if typical queries are highly selective, the absence of a result ranking as in the Boolean retrieval model is not a problem. Nowadays, though, document collections (e.g., the Web) are large, so that even selective queries yield hundreds if not thousands of result documents. Having to sift through all of them is clearly unacceptable for the user. Instead, result documents should be ranked and presented to the user in the order of their believed relevance. The vector space model (VSM) proposed by Salton et al. [SWY75] applies this idea and supports a more fine-grained assessment of relevance than the Boolean retrieval model.

The VSM models documents and queries as $|\mathcal{V}|$ -dimensional vectors, i.e., there is a component in the vector for each term in the vocabulary. Let \vec{q} and \vec{d} be the vectors corresponding to a query q and a document d , respectively. In order to assess the document's relevance to the query, the VSM compares the directions of their corresponding vectors. One established measure to do so is the cosine similarity. For vectors that point in the same direction and thus have a 0° angle between them, it assumes its maximal value 1. The cosine similarity is formally defined as follows:

Definition 2.1 (Cosine similarity) Let \vec{q} and \vec{d} be two document vectors. The cosine similarity, indicating the cosine of the angle between \vec{q} and \vec{d} , is

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}||\vec{d}|}, \quad (2.1)$$

where $\vec{q} \cdot \vec{d}$ is the dot product between the two vectors and $|\vec{q}|$ and $|\vec{d}|$ are the vectors' Euclidean lengths.

TF-IDF Weighting

Our description of the VSM has not yet addressed the question how the components' values are determined. In analogy to the Boolean retrieval model, one could deal with only binary vectors, setting their components to either 1 or 0 depending on whether the term is present or not in the document. Notice, however, that this throws away the crucial information about how often the term occurs in the document. To take this into account, documents need to be modeled in a way that preserves this information. The *bag of words* model views each document as a bag (i.e., multiset) of terms. We let $tf_{v,d}$ denote the *term frequency* of term v in the document d . Having preserved term frequency information, one could set $\vec{d}(v) = tf_{v,d}$. One problem with this approach is that it implicitly assigns equal importance to all terms when assessing the relevance of a document. To see why this is problematic, consider the query *white orchid*. It is natural to expect that the term *white* occurs in many more documents than the term *orchid* – the term *orchid* is thus more *discriminative*. The relative weight of query terms that are less discriminative, such as *white*, should be reduced. Let df_v denote the *document frequency* of term v , i.e., the number of documents that contain v . We formally define the *inverse document frequency* of v as

$$idf_v = \log \frac{|\mathcal{D}|}{df_v}. \quad (2.2)$$

One way to determine components in the VSM combining the two is to set

$$\vec{d}(v) = \text{tf}_{v,d} \cdot \text{idf}_v . \quad (2.3)$$

The above is only one example of a basic *tf-idf* weighting scheme. Over the years, many refinements and extensions of this general idea have emerged – for a more detailed discussion of these we refer to Manning et al. [MRS08].

Okapi BM25

Okapi BM25 can be regarded as another *tf-idf* weighting scheme. However, it differs from the ones discussed above in one important aspect. Okapi BM25 can be seen as an extension of the *binary independence model* in *probabilistic information retrieval* and has thus a well-grounded theoretical foundation. Even more important in practice, it has repeatedly been shown to achieve excellent retrieval effectiveness, i.e., produce highly relevant result documents, which makes it one of the current state-of-the-art methods. Our following description will not go into detail on the connection between probabilistic information retrieval and Okapi BM25 – for a detailed discussion of this aspect we refer to Spärck Jones et al. [JWR00a, JWR00b].

Definition 2.2 (Okapi BM25) *Let d be a document and q be a keyword query, Okapi BM25 defines the document's relevance to the query as*

$$r(q, d) = \sum_{v \in q} w_{\text{idf}}(v) \cdot w_{\text{tf}}(v, d) . \quad (2.4)$$

The weight $w_{\text{idf}}(v)$ reflects the inverse document frequency of term v in the collection and is defined as

$$w_{\text{idf}}(v) = \log \frac{N - \text{df}_v + 0.5}{\text{df}_v + 0.5} , \quad (2.5)$$

where N and df_v have the aforementioned semantics.

The weight $w_{\text{tf}}(v, d)$ reflects the frequency of term v in document d and is defined as

$$w_{\text{tf}}(v, d) = \frac{(k_1 + 1) \cdot \text{tf}_{v,d}}{k_1 \cdot ((1 - b) + b \cdot (|d|/\text{avdl})) + \text{tf}_{v,d}} , \quad (2.6)$$

where $0 \leq b \leq 1$ and $k_1 \geq 0$ are tunable parameters.

Okapi BM25 thus takes into account the plain term frequency $\text{tf}_{v,d}$ but scales term frequencies based on the ratio between the document's length $|d|$ and the

average document length $avdl$ observed in the document collection. The term frequency scaling can be adjusted by means of the parameters k_1 and b . The parameter k_1 controls the impact of the magnitude of the frequency. The choice $k_1 = 0$, for instance, yields binary values of $w_{tf}(v, d)$ indicating whether the term is present or not. The second parameter b controls the strength of document length normalization. Choices of b close to 1 thus put more penalty on long documents by discounting their term frequencies. Typical choices of the two parameters that have been proven to work in practice are $k_1 = 1.2$ and $b = 0.75$.

Language Models

Language models have been used in other areas of computer science, such as speech recognition. In general, a language model is a probabilistic model that generates outputs. The corresponding set of possible outputs is referred to as the language of the model. Ponte and Croft [PC98], more than a decade ago, were the first to apply language models to information retrieval. Language-modeling approaches to information retrieval associate with each document in the collection a language model. In the so-called *query-likelihood approach*, the probability $P(q|d)$ of generating the query q from the language model associated with document d is used to rank query results.

The *unigram language model* is one common type of language model that bears some resemblance with the *tf-idf* weighting schemes discussed above. Again, documents are considered as bags of words – the order of terms is disregarded. The language model associated with document d generates the term v with probability $P(v | d)$ which is estimated as

$$P(v | d) = \frac{tf_{v,d}}{|d|}, \quad (2.7)$$

thus being estimated as the term's relative term frequency in the document.

Given a keyword query q , the probability $P(q|d)$ of generating the query from the language model associated with document d is estimated as

$$P(q | d) = \prod_{v \in q} P(v | d). \quad (2.8)$$

By this definition, the probability $P(q | d)$ is zero, if the document does not contain all query terms. This may often be too restrictive and rule out many

nevertheless relevant documents. *Smoothing techniques* address this zero-probability problem, for instance, by estimating the probabilities not only based on the document itself, but taking into account the whole document collection. Jelinek-Mercer smoothing, as one popular smoothing technique, employs a *linear interpolation* when estimating the probability $P(v | d)$. Let

$$cf_v = \sum_{d \in D} tf_{v,d} \quad (2.9)$$

denote the *collection frequency* of term v and $\lambda \in [0, 1]$ be a mixture parameter, then a smoothed estimate for the probability of generating the term v from document d is

$$P(v | d) = (1 - \gamma) \cdot \frac{cf_{v,d}}{\sum_{v \in V} cf_{v,d}} + \gamma \cdot \frac{tf_{v,d}}{|d|} . \quad (2.10)$$

The second term regards the entire document collection as a single document. Zhai and Lafferty [ZL04] give an overview of existing smoothing approaches and compare them empirically.

Smoothing has additional subtler effects beyond addressing the zero-probability problem mentioned above. In detail, smoothing achieves an effect similar to the weighting by inverse document frequencies in other retrieval models, since it reduces the impact of less discriminative terms.

Phrase Queries & Proximity

The models discussed so far disregard the order of terms in queries and documents. Phrase queries ask for documents that contain the query terms in the same order as they appear in the query. Techniques for the efficient evaluation of phrase queries were proposed by Chang and Poon [CP08] and Williams et al. [WZB04]; their impact on retrieval effectiveness was studied by Croft et al. [CTL91], Mitra et al. [MBSC97], and most recently Broschart et al [BBS10].

Phrase queries filter out documents that do not contain the given query phrase. Proximity scoring techniques, as less rigid approaches, score a document based on how closely together query terms occur in it. Büttcher et al. [BCL06], Tao and Zhai [TZ07], as well as Zhao and Yeogirl [ZY09] propose and compare different proximity measures. Efficient early-terminating query-processing techniques that support proximity scoring are proposed by Schenkel et al [SBwH⁺07] and Zhu et al. [ZSLW09].

In practice, as a final remark, the separation between retrieval models is less strict than our preceding discussion. Thus, one may combine Boolean retrieval and Okapi BM25 to rank result documents according to their relevance score but filter out those that do not match the Boolean query. Under so-called *conjunctive query semantics*, as one variant of this combination, the user’s query is implicitly treated as a conjunction of query terms – the result thus consists of documents that contain all query terms in descending order of their relevance score. In an analogous manner, phrase queries, as described above, and relevance scoring (e.g., using Okapi BM25) can be combined.

2.1.2 Link Analysis

The relevance score determined by a retrieval model, such as those discussed above, is only one of the signals that modern search engines use to rank query results. For hyperlinked corpora, such as the Web, additional clues for the result ranking can be obtained from link analysis techniques. HITS [Kle99] and PageRank [PBMW98] are among the earliest and still most popular techniques. PageRank, which was a crucial building block in the original Google [BP98] search engine, views the collection of hyperlinked documents as a directed graph $G(V, E)$ with vertices V corresponding to documents and edges E corresponding to hyperlinks between the documents. On this graph, PageRank scores that reflect the importance of web pages are computed based on the following definition:

Definition 2.3 (PageRank) *Let $G(V, E)$ be a directed graph. The PageRank score $r(v)$ of a node v is defined recursively based on nodes that point to v as*

$$r(v) = (1 - \epsilon) \cdot \sum_{(u,v) \in E} \frac{r(u)}{\text{out}(u)} + \frac{\epsilon}{|V|}, \quad (2.11)$$

where $\text{out}(u)$ is the outdegree of the node u and ϵ is a tunable parameter.

The above equation describes a random walk on the graph $G(V, E)$. In each step, with probability $1 - \epsilon$, one of the outgoing edges of the current node is chosen with uniform probability and traversed, as captured in the first part of the equation. Otherwise, with probability ϵ , a so-called random jump is performed to any node in the graph chosen uniformly at random, as captured in the second part of the equation. The PageRank scores $r(v)$ are the stationary

visiting probabilities of this random walk, corresponding to the fraction of time that is spent in a node as the random walk continues to infinity. It can be shown that the random walk corresponds to an ergodic Markov chain, which in turns guarantees existence and uniqueness of the stationary visiting probabilities.

Many extensions and refinements of PageRank and HITS have been proposed with underlying objectives including, for instance, the detection of spam web pages [CDG⁺07], as well as topic-specific [Hav02] and time-aware authority ranking [BVW05, BBVW06].

2.1.3 Indexing & Compression

Assessing the relevance of documents to an information need is an important problem in information retrieval, and a good solution to it is crucial to having an effective information retrieval system. In practice, though, achieving quick response times is equally important. Clever indexing of the document collection is key toward achieving this objective. This section gives an overview of common indexing, compression, and query-processing techniques.

Inverted File Index

The *inverted file index* [ZM06] (often referred to as *inverted index* for short) is the workhorse of information retrieval and the foundation of many real-world information retrieval system including all of today's major web search engines. Like the index in the backmatter of a book, the inverted index keeps track of where a term occurs in the document collection. Conceptually, an inverted file index consists of two components, namely a (i) *lexicon* containing all terms (ii) one *posting lists* per term in the lexicon with information about the term's occurrences in the document collection. Figure 2.1 shows an example inverted index with posting lists for the terms *cat* and *dog*.

The lexicon is often kept in main memory, organized, for instance, using a hash table. When kept on hard disk and accessed from there, the lexicon can be organized using a B-Tree [Com79]. A detailed discussion of different lexicon implementations and their respective trade-offs is given in Witten et al. [WMB99].

Posting lists typically reside on hard disk and are fetched at query-processing time. The posting list L_v for the term $v \in \mathcal{V}$ consists of postings having the form

$$(d, p) ,$$

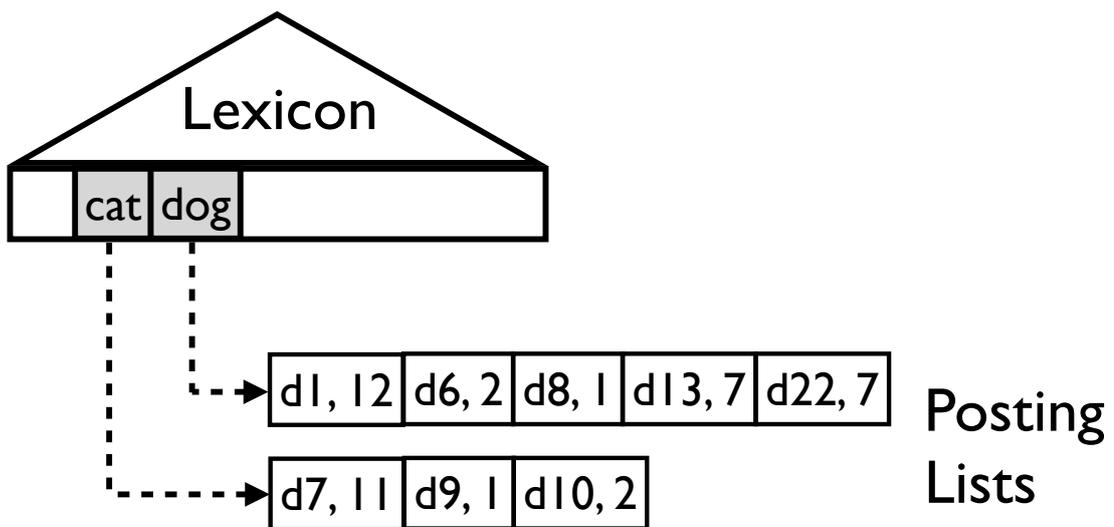


Figure 2.1: Components of an inverted index

where d is a document identifier and p is a payload whose specifics depend on which query types must be supported efficiently.

In the simplest case, if only queries according to the Boolean retrieval model must be supported, the payload remains empty. For ranked retrieval models, such as Okapi BM25, scalar payloads are necessary. These scalar payloads may contain the (i) term frequency $tf_{v,d}$ (like in our example inverted index shown in Figure 2.1) or a (ii) precomputed $tf_{v,d} \cdot idf_v$ score contribution. Finally, if phrase queries must be supported, payloads keep track of the positions where the term occurs in the document. If more than one these query types should be supported efficiently, payloads can be extended to accommodate all required information (e.g., both term frequency and positional information). Alternatively, for each term, multiple posting lists, namely one per query type, can be kept. The trade-offs of these two schemes are investigated by Anh and Moffat et al. [AM06c].

Posting lists can either be *document-ordered*, i.e., the postings are sorted in ascending order of their document identifier. Or, for scalar payloads, posting lists can be sorted in descending order of these payloads, which is then referred to as *frequency-ordered* or *score-ordered* posting lists. As we detail below, document-sorted posting lists allow for better compression. Frequency-ordered postings lists, on the other hand, allow for query-processing techniques that terminate early once the k most relevant documents have been identified and thus do not read posting lists in their entirety.

Compression

Apart from the obvious storage savings, compression of posting lists is beneficial for several other reasons. One is that performance of CPUs has increased relatively more in recent years than the performance of hard disks. As a consequence, as Manning et al. [MRS08] report, reading the compressed posting list from disk and decompressing it is often faster than reading the same posting list in uncompressed form.

Compression of inverted file indexes is a well-studied research topic. The techniques that we discuss next were chosen so as to demonstrate some recurring ideas. For a more comprehensive overview of applicable techniques, we refer to Witten et al. [WMB99], as well as Zobel and Moffat [ZM06].

When implementing an inverted index in a naïve manner, one may choose to represent and store all integer values using the same amount of bits (e.g., 32 or 64). The key idea behind the compression techniques presented next is to represent and store smaller values using fewer bits. This is particularly promising if many of the values are expected to be small as, for instance, for term frequencies.

Unary encoding represents a positive integer as $x - 1$ one bits followed by a single zero bit. As a concrete example, $x = 13$ is encoded as

$$\langle 1111111111110 \rangle .$$

Unary encoding obviously only saves space for small x .

Elias- γ encoding represents a positive integer x using two parts. The first part represents the value $1 + \lfloor \log x \rfloor$ using the aforementioned unary encoding. The second part encodes $x - 2^{\lfloor \log x \rfloor}$ in $\lfloor \log x \rfloor$ bits using standard binary encoding. Our example value $x = 13$ would thus be represented as

$$\langle 1110 | 101 \rangle .$$

Elias- δ encoding differs from Elias- γ encoding in representing the first part, i.e., the value $1 + \lfloor \log x \rfloor$ using Elias- γ encoding instead of unary encoding. Hence, $x = 13$ would be encoded as

$$\langle 110 | 00 | 101 \rangle ,$$

where the first two parts represent $4 = 1 + \lfloor \log 13 \rfloor$.

7-Bit encoding, in contrast to the preceding encodings, operates at the byte granularity. The representation produced is thus always byte-aligned, using an

integral number of bytes, which allows for more efficient decompression. 7-Bit encoding, as the name suggests, uses only the last seven bits per byte to encode the original information; the first (i.e., most significant) bit in each byte serves as a continuation bit. Given a non-negative integer x , 7-Bit encoding uses groups of seven bits from x 's binary representation starting from the least significant. One byte is output for each of these groups with the most significant bit set if more bytes follow and not set if this byte contained the most significant bit from x 's binary representation. As a concrete example, the value $x = 131$ would be encoded as

$$\langle 00000001 \mid 10110001 \rangle .$$

Gap encoding, as the last technique discussed here, operates on an ascending sequence of m integers

$$\langle x_1, \dots, x_m \rangle .$$

These could be, for example, the positions where a term occurs in a document (i.e., the payload needed to support phrase queries as detailed above) or the sequence of document identifiers in a document-sorted posting list. Gap encoding transforms the integer sequence into

$$\langle x_1, x_2 - x_1, \dots, x_m - x_{m-1} \rangle$$

by encoding all but the first integer as the difference to its predecessor. In doing so it yields smaller values that can then be represented more compactly using one of the above encoding techniques.

2.1.4 Query Processing

So far, our focus has been on assessing the relevance of a document to a query and on indexing the document collection to quickly find out about occurrences of a particular term. One question still left open is how we actually process a given query q , i.e., determine documents considered highly relevant using the index created on the document collection. To answer this question, we next describe three common query-processing techniques.

Term-at-a-Time

Term-at-a-time query processing reads the posting lists L_v for $v \in q$ one after the other – hence the name *term at a time*. For each candidate document d discovered, an *accumulator* is kept in main memory that keeps the partial relevance

score accumulated so far for this document. When the posting lists for all query terms have been completely read, the set of candidates is filtered (e.g., to remove documents that do not contain all query terms), sorted in descending score order, and output.

Depending on specific requirements, term-at-a-time query processing can be further improved. If, for instance, conjunctive query semantics is desired (i.e., all query terms must appear in a document to be reported as a result), posting lists can be processed in ascending order of their length. Additional accumulators have to be initialized only while reading the first posting list, since documents that are not contained in the first posting list cannot contain all query terms. The required amount of main memory then depends on the length of the shortest posting list, i.e., the most selective query term.

Document-at-a-Time

Document-at-a-time query processing reads and merges the document-ordered posting lists for all $v \in q$ in parallel, similar to a merge join in a database system [RG03]. It is geared at a conjunctive query semantics, i.e., all query terms must be present in a document. The method merges the posting lists thereby exploiting their common sort order. To this end, the method maintains a cursor for each postings list and advances in each round the cursor pointing to the smallest document identifier, which can be implemented efficiently using a priority queue. When all cursors point to postings belonging to the same document, the document is evaluated, i.e., its score is computed and additional checks (e.g., if the document contains a given query phrase) may be performed. Following that, the document is added to the set of result documents, which is organized using a priority queue.

Early-Terminating Methods

Our presentation of the term-at-a-time and document-at-a-time query processing methods above assumed that we are interested in a holistic evaluation of the query, i.e., in computing all query results. Often, however, it is sufficient to compute only the top-k results, e.g., because these are the results the user is likely to look at. Ideally, one would like to avoid reading posting lists in their entirety, but have a criterion to terminate query processing early, once the best k results have been identified. Although originally not designed for IR, the NRA (an ab-

abbreviation for no random accesses) algorithm proposed by Fagin et al [FLN03] is one method that allows early termination.

NRA reads score-ordered posting lists for all $v \in q$ in parallel. Candidate result documents are managed using a hash table C . For each candidate result document d NRA maintains a lower bound $worst(d)$ and an upper bound $best(d)$ on its relevance score, respectively. The lower bound $worst(d)$ is based on already-seen postings for document d . The upper bound $best(d)$ leverages the fact that posting lists are score-ordered and takes into account the currently-seen score per posting list as an upper bound for the scores of postings yet to read from this list. Using the same rationale one can define $bestUnseen$ as the best possible relevance that a still unseen result document can achieve. The k candidate result documents having the highest lower bound are additionally kept in a priority queue $topK$. NRA lets min_k denote the lower bound of the minimal candidate result document in this priority queue. While reading posting lists in parallel, NRA incrementally maintains the lower bound and upper bound per candidate result document, as well as the values of min_k and $bestUnseen$.

Once the following conditions hold, NRA safely terminates:

- (i) at least k candidate result documents have been seen, i.e.,

$$|C| \geq k$$

- (ii) no unseen result document can make it into the top- k , i.e.,

$$bestUnseen \leq min_k$$

- (iii) there is no seen candidate result documents that still has the potential to make it into the top- k , i.e.,

$$\forall d \in C \setminus topK : best(d) \leq min_k .$$

When NRA stops, it is guaranteed to have identified a correct top- k result. The relative order of result documents in the identified top- k may however deviate from their true relative order.

We chose to present NRA because of its versatility – it is applicable for a large class of aggregation functions. Note that other early-terminating methods have been proposed in the IR literature, among them work by Turtle and Flood [TF95], Buckley [BL85], and Anh and Moffat [AM06a, AM06b].

2.1.5 Evaluation

Having described important building blocks of an information retrieval system, we now turn our attention to how such a system can be evaluated with regard to its retrieval effectiveness. The measures described below can be divided into (i) measures that assess the quality of a query result based on a ground truth or relevance assessments, (ii) measures that compare two query results, and (iii) a measure of agreement among users assessing the quality of query results.

Precision and Recall

We next introduce *precision* and *recall*, as the best-known evaluation measures in IR, and their variations that only consider a prefix of the query result.

Definition 2.4 (Precision) Let $R = \langle r_1, \dots, r_m \rangle$ with $r_i \in \mathcal{D}$ be a query result and let $G \subseteq \mathcal{D}$ denote the ground truth set of relevant documents. The precision achieved by the query result R is given as

$$\text{Precision}(R) = \frac{|R \cap G|}{|R|} \quad (2.12)$$

and reflects how many of the returned result documents are actually relevant.

Definition 2.5 (Recall) Let $R = \langle r_1, \dots, r_m \rangle$ with $r_i \in \mathcal{D}$ be a query result and let $G \subseteq \mathcal{D}$ denote the ground truth set of relevant documents. The recall achieved by the query result R is given as

$$\text{Recall}(R) = \frac{|R \cap G|}{|G|} \quad (2.13)$$

and reflects how many of the known relevant documents are returned.

Our description thus far considered the entire query result R . It is often more appropriate to consider only the top- k result $\langle r_1, \dots, r_k \rangle$, for instance, because these are the results that the user is likely to inspect. Therefore, we define variants of the above measures at cut-off level k that are defined in exactly the same way, but only consider the top- k query result $\langle r_1, \dots, r_k \rangle$. These are further referred to as precision at k ($P@k$) and recall at k ($R@k$), respectively.

Normalized Discounted Cumulative Gain (nDCG)

Precision and recall do not take into account the ranking of query results, i.e., it only matters *whether* relevant documents are retrieved but not *where* they are

reported in the query result. Apart from that, they assume binary relevance assessments, i.e., a document is either relevant or non-relevant to a query. This is restrictive for two reasons. First, when conducting a user study, users may have different opinions on the relevance of a document; to apply precision and recall these would have to be aggregated into a single binary relevance assessment (e.g., by applying a majority voting scheme). Second, it is often desirable to allow for a more fine-grained grading of relevance using grades like 0 (*non-relevant*), 1 (*marginally relevant*), and 2 (*relevant*). Normalized discounted cumulative gain (nDCG), proposed by Järvelin and Kekäläinen [JK02], is a more recent measure that takes into account these two points.

Definition 2.6 (Normalized discounted cumulative gain) *Let R be a query result and $0 \leq G(r_i) \leq G_{\max}$ denote a grade given by users to the result document r_i . The normalized discounted cumulative gain at cut-off level k is then defined as*

$$\text{nDCG}(R, k) = Z_k \cdot \sum_{i=1}^k \frac{2^{G(r_i)} - 1}{\log(1 + m)} \quad (2.14)$$

with a normalization constant Z_k determined so that $\text{nDCG}(R^*, k) = 1$ for an ideal result R^* that returns the k best result documents in descending order of their grades. This normalization ensures that nDCG values are in $[0, 1]$.

We now proceed to the second group of measures that compare two query results R and R' . In the following chapters, these measures will be used to evaluate approximate techniques, i.e., R' will be an approximation of R whose accuracy we would like to assess.

Relative Recall

Relative recall is defined in analogy to the definition of recall above, but assumes the ground truth G to be the original query result R . It thus reflects how many of the original query results are contained in the approximate query result R' . Apart from that, it is also possible to compare only the top- k original and approximate query results, which is then referred to as relative recall at cut-off level k ($\text{RR}@k$).

Kendall's τ

Kendall's τ was originally proposed to compare two permutations and measure their mutual agreement in order. Among several slight variations of Kendall's τ ,

that can be found in the literature, we adopt the following definition given in Boldi et al. [BSV05]:

Definition 2.7 (Kendall's τ) Let R and R' be two query results that contain the same set of documents $\{d_1, \dots, d_m\}$ potentially in different order. Further, we let $\text{pos}(d_i, R)$ and $\text{pos}(d_i, R')$ denote the position of result document d_i in R and R' , respectively. We call a result document pair (d_i, d_j)

- *concordant*, iff $(\text{pos}(d_i, R) - \text{pos}(d_j, R)) \cdot (\text{pos}(d_i, R') - \text{pos}(d_j, R')) > 0$.
- *discordant*, iff $(\text{pos}(d_i, R) - \text{pos}(d_j, R)) \cdot (\text{pos}(d_i, R') - \text{pos}(d_j, R')) < 0$.

Let $C(R, R')$ and $D(R, R')$ denote the set of concordant and discordant result document pairs, respectively. Kendall's τ between R and R' is given as

$$\tau(R, R') = \frac{C(R, R') - D(R, R')}{C(R, R') + D(R, R')} \quad (2.15)$$

The value of Kendall's τ is in $[-1, 1]$, where 1 signals a perfect agreement between R and R' on the order of result documents. Conversely, a value of -1 signals that R is the reverse of R' and that the two thus maximally disagree on the order of result documents.

Fleiss' κ

The κ statistic proposed by Fleiss [Fle71], as the last measure discussed in this section, measures the agreement among a set of raters, i.e., in our case users who assess the quality of result documents. Fleiss' κ assumes that there is a fixed number n of users (e.g., five) per item (i.e., a pair consisting of a query and a result document) who assess the relevance of the result document to the query – each of the m items may be assessed by a different set of users. Quality assessments are made by assigning an item to one of k categories (e.g., grades).

Definition 2.8 (Fleiss' κ) Let m be the number of items, n be the number of users who assess each item, k be the number of categories, and $g_{i,j}$ (with $1 \leq i \leq m$ and $1 \leq j \leq k$) denote the number of users who assigned category j to item i . The proportion p_j of assignments to the j -th category is

$$p_j = \frac{1}{m \cdot n} \cdot \sum_{i=1}^m g_{i,j} \quad (2.16)$$

The extent to which users' assessments agree for the i -th item is

$$P_i = \frac{1}{n \cdot (n - 1)} \cdot \sum_{j=1}^k g_{i,j} \cdot (g_{i,j} - 1), \quad (2.17)$$

which is maximal, assuming a value of 1, if all user assessments agree on a category for the i -th item. Let

$$\hat{P} = \frac{1}{m} \cdot \sum_{i=1}^m P_i \quad (2.18)$$

denote the mean agreement of users' assessments. Further, let \hat{P}_e be defined as

$$\hat{P}_e = \sum_{j=1}^k p_j^2 \quad (2.19)$$

reflecting the agreement that would be expected if assessments were done randomly. Fleiss' κ statistic is then defined as

$$\kappa = \frac{\hat{P} - \hat{P}_e}{1 - \hat{P}_e}. \quad (2.20)$$

Fleiss' κ thus measures how much the observed agreement among users' assessments deviates from the agreement that would be obtained for random assessments. If there is complete agreement among users, the statistic yields a value $\kappa = 1$. If there is poor or no agreement among users, a value $\kappa < 0$ is yielded.

The notation introduced in our brief overview of information retrieval is summarized in Table 2.1. For more comprehensive introductions to IR we refer to the textbooks by Manning et al. [MRS08], Baeza-Yates and Ribeiro-Neto [BYRN99], as well as Chakrabarti [Cha02]. Witten et al. [WMB99] and Croft et al. [CMS09], finally, are good sources to find more details about implementation subtleties.

2.2 Natural Language Processing

Natural language processing (NLP) as an academic field is concerned with automatically processing, understanding, and analyzing human language (e.g., English, German, or Swedish) as opposed to computer languages (e.g., Java, Fortran, or C++). Giving a complete overview of the field is beyond the scope of this work and we refer the reader to the textbook by Manning and Schütze [MS99]

| | |
|----------------|-----------------------------------------------------------------------------------------|
| \mathcal{V} | Vocabulary |
| \mathcal{D} | Document collection |
| $ d $ | Length of document d |
| $avdl$ | Average document length |
| N | Size of the document collection |
| $tf_{v,d}$ | Frequency of term v in document d |
| cf_v | Frequency of term v in the document collection |
| df_v | Document frequency |
| $Precision(R)$ | Precision achieved by query result R |
| $Recall(R)$ | Recall achieved by query result R |
| $P@k(R)$ | Precision at cut-off level k achieved by query result R |
| $R@k(R)$ | Recall at cut-off level k achieved by query result R |
| $nDCG(R, k)$ | Normalized discounted cumulative gain at cut-off level k achieved by query result R |
| $RR(R, R')$ | Relative recall of R' w.r.t. R |
| $RR@k(R, R')$ | Relative recall at cut-off level k of R' w.r.t. R |
| $\tau(R, R')$ | Kendall's τ |
| κ | Fleiss' κ |

Table 2.1: Summary of notation

for a good set of entry points into the field. We limit our attention to presenting few ideas and techniques originating from natural language processing that our work builds upon.

2.2.1 Semantic Similarity

Given two words v and w , we may ask how similar they are in meaning. This question is of interest in a number of applications. In *synonymy detection*, as a first application, one is interested in finding synonyms of a given word (e.g., car) or detecting whether two words (e.g., car and automobile) are synonyms. *Query-expansion techniques in information retrieval*, as a second application, automatically augment the user's query with words that have similar meaning to improve the quality of query results. What is needed, is a measure that assesses the degree of semantic similarity between the two given words v and w . However, as Manning and Schütze [MS99] point out, there is no generally accepted notion of semantic similarity. Some approaches proposed in the literature con-

sider two terms semantically similar if they are *near-synonyms* (e.g., *cab* and *taxi*); other approaches demand that the two words are likely to co-occur (e.g., *car* and *engine*). Our focus will be on the former interpretation.

Strong Contextual Hypothesis

How can we measure the degree of semantic similarity between two words? The key observation here is that semantically similar words tend to occur in similar contexts. For instance, the words *car* and *automobile* from our earlier example word pair occur frequently together with words such as *fuel*, *street*, or *engine*. Thus, by comparing the two words' contexts (i.e., their frequently co-occurring words), we should obtain a good estimate of their semantic similarity. This idea has existed in the literature for more than four decades, and was first proposed, to the best of our knowledge, by Rubenstein and Goodenough [RG65]. Miller and Charles [MC91] cast the idea into the following *strong contextual hypothesis*:

Two words are semantically similar to the extent that their contextual representations are similar.

Comparing Word Contexts

When implementing the above idea, we need a formal definition of *word context* and a *similarity measure* between word contexts. Various concrete implementations have been proposed, two of them are sketched briefly in the following.

One simple way of defining the context of v and w , is to consider the sets of all terms $C(v)$ and $C(w)$ that co-occur with v and w (e.g., within a sentence) in our document collection, respectively. These sets can then be compared using a set-based similarity measure such as the Jaccard coefficient, yielding

$$\text{sim}(v, w) = \frac{|C(v) \cap C(w)|}{|C(v) \cup C(w)|} \quad (2.21)$$

as a measure of semantic similarity.

This definition disregards frequency information entirely. In analogy to the Vector Space Model described earlier, we can represent the context of a word w as a \mathcal{V} -dimensional vector. Let $\text{cooc}(v, w)$ denote the number of times that w co-occurs with v in the document collection. We can set the component $\vec{v}(w) = \text{cooc}(v, w)$ and compare the resulting word context vectors using the cosine similarity discussed above.

An alternative information-theoretic approach that also takes into account frequency information is based on Kullback-Leibler (KL) divergence, that is also known as relative entropy, and defined as follows:

Definition 2.9 (Kullback-Leibler divergence) *Given two probability mass functions $v(z)$ and $w(z)$ their Kullback-Leibler (KL) divergence is*

$$D(v||w) = \sum_{z \in \mathcal{V}} v(z) \cdot \log \frac{v(z)}{w(z)}. \quad (2.22)$$

Let $\text{occ}(v)$ denote the number of times that v occurs in the document collection, we can define the two probability mass functions

$$v(z) = \frac{\text{cooc}(v, z)}{\text{occ}(v)} \quad \text{respectively} \quad w(z) = \frac{\text{cooc}(w, z)}{\text{occ}(w)} \quad (2.23)$$

and apply KL divergence as described above.

One idea that has attracted interest in recent years is to use the Web, as the arguably largest available corpus, to determine word contexts and thus measure the semantic similarity between words. Two representative approaches, both employing a major web search engine as a proxy to obtain word contexts, are Sahami and Heilman [SH06] and Cilibrasi and Vitanyi [CV07].

2.2.2 Hidden Markov Models

Hidden Markov Models (HMMs) are a powerful class of statistical models with many applications including *speech recognition*, *statistical machine translation*, and *part-of-speech tagging*. Underlying a HMM is a Markov process that models an aspect of the real world. We only consider first-order HMMs here, for which the underlying Markov process can be thought of as a finite-state machine that has probabilistic state transitions and starting states, and outputs a random symbol whenever entering a state.

Definition 2.10 (Hidden Markov Model) *A (first-order) HMM consists of (i) a state space $S = \{s_1, \dots, s_n\}$, (ii) an alphabet of output symbols Σ , (iii) initial state probabilities with $P(s_i)$ as the probability that the process starts from state s_i , (iv) transition probabilities with $P(s_j | s_i)$ as the probability that the process moves from state s_i to state s_j , and (v) output probabilities with $P(\sigma | s_i)$ as the probability that the process outputs symbol σ when in state s_i .*

Example

Figure 2.2 shows an example first-order HMM. This toy HMM models a scenario where one can only observe the weather, but has lost track of whether it's a weekend day or working day. The state space S of our HMM comprises two states that correspond to weekend days (WE) and working days (WD). The alphabet Σ consists of two output symbols Sun and Rain that reflect the weather on a particular day. Our HMM starts with equal probability from state WD or state WE. Further, the probability of having sun is slightly larger on a weekend, whereas on a working day the probability of having rain is slightly larger.

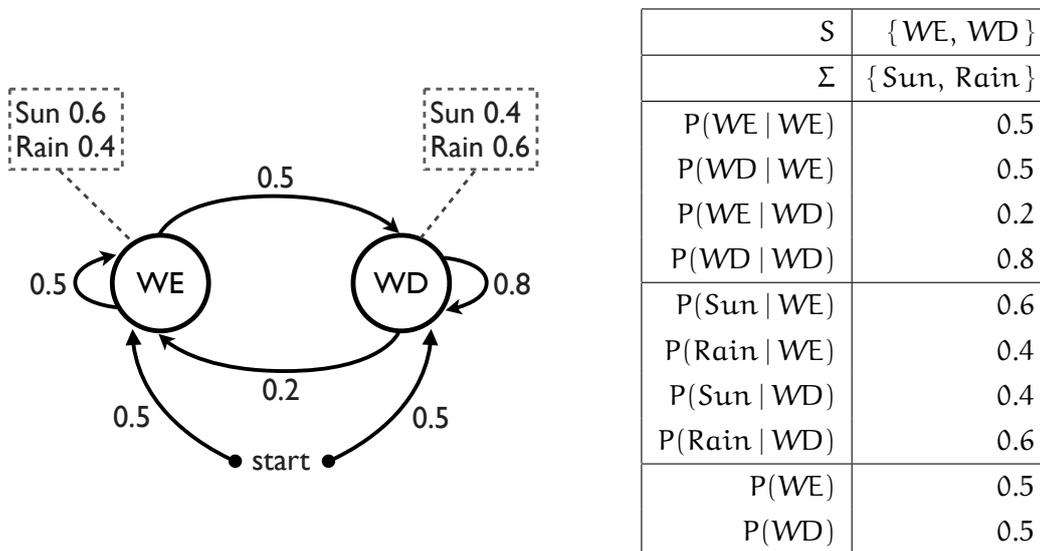


Figure 2.2: Example Hidden Markov Model

The Viterbi Algorithm

Given a HMM and an observed output $\langle \sigma_1, \dots, \sigma_m \rangle$, one is often interested in the most plausible explanation of the observed output, i.e., the state sequence having the highest probability of being traversed while producing the observed output. Assume that we have observed the weather sequence $\langle Sun, Rain, Sun \rangle$ during the past three days for our above example HMM. By finding the state sequence that is most likely of having produced this weather sequence, we get an idea about which of the past three days were weekend days or working days. Let $\delta(i, j)$ denote the maximum probability of being in state s_i having output the prefix $\langle \sigma_1, \dots, \sigma_j \rangle$ of the observed output. For $j = 1$ we initialize $\delta(i, 1)$ as $P(s_i) \cdot P(\sigma_1 | s_i)$, i.e., the probability of starting from state s_i and producing the

symbol σ_1 . For $j > 1$ the value $\delta(i, j)$ can be computed using the recurrence

$$\delta(i, j) = \max_{1 \leq k \leq n} \delta(k, j-1) \cdot P(s_i | s_k) \cdot P(\sigma_j | s_k). \quad (2.24)$$

The rationale behind this recurrence is that the maximum probability state sequence that ends in s_i and produces the prefix $\langle \sigma_1, \dots, \sigma_j \rangle$ can be determined as the best extension of any state sequences producing the prefix $\langle \sigma_1, \dots, \sigma_{j-1} \rangle$.

The Viterbi algorithm determines the maximum probability state sequence using dynamic programming. Pseudo code of the method is given in Algorithm 1.

The algorithm computes values $\delta[i][j]$ incrementally based on the above recurrence. When computing $\delta[i][j]$ for $j > 1$ the maximizing value of k in the above recurrence is kept in $\psi[i][j]$. This value thus keeps track of s_i 's predecessor on the best state sequence that outputs $\langle \sigma_1, \dots, \sigma_j \rangle$. In the backtracking phase, the algorithm leverages this information to assemble the best state sequence that produces the observed output. The space complexity of the Viterbi algorithm is in $O(n \cdot m)$ for memoizing the dynamic programming tables $\delta[i][j]$ and $\psi[i][j]$. Its time complexity is in $O(n^2 \cdot m)$ – for each prefix of the observed output all state pairs need to be examined.

For more comprehensive introductions to HMMs, we refer to the survey by Rabiner [Rab90] and the textbook by Manning and Schütze [MS99]. The textbook by Allen [All72] is a good reference to find out about Markov processes in general.

2.2.3 Temporal Information Extraction

Extracting temporal information such as *temporal expressions* and *event expressions* from text documents is important to obtain a temporal interpretation of the documents' contents. Later, in Chapter 5, we describe techniques that make use of temporal expressions (e.g., “January 15, 2009”, “last week”, and “in 2009”). In this section, we provide a brief overview of temporal expressions and existing approaches for their extraction and annotation.

Classes of Temporal Expressions

Alonso et al. [AGBY07] distinguish three classes of temporal expressions:

- *Explicit*. These include temporal expressions such as “January 5, 2009”, “December 1996”, or “in 1945” that have an immediate interpretation.

Algorithm 1: Viterbi algorithm

Data: HMM and observed output $\{\sigma_1, \dots, \sigma_m\}$ **Result:** Maximum probability state sequence ρ

```

1  $\delta[1..n][1..m]$  // Maximum probabilities  $\delta(i, j)$ 
2  $\psi[1..n][1..m]$  // Predecessors
3  $\rho[1..m] = \langle 1, \dots, 1 \rangle$  // Maximum probability state sequence  $\rho$ 
4 /* Initialization */
5 for  $i = 1 \dots n$  do
6    $\delta[i][1] = P(s_i) \cdot P(\sigma_1 | s_i)$ 
7 /* Dynamic Programming */
8 for  $j = 2 \dots m$  do
9   for  $i = 1 \dots n$  do
10    for  $k = 1 \dots n$  do
11     if  $\delta[k][j - 1] \cdot P(s_i | s_k) \cdot P(\sigma_j | s_i) > \delta[i][j]$  then
12       $\delta[i][j] = \delta[k][j - 1] \cdot P(s_i | s_k) \cdot P(\sigma_j | s_i)$ 
13       $\psi[i][j] = k$ 
14 /* Backtracking */
15 for  $i = 1 \dots n$  do
16   if  $\delta[m][i] > \delta[m][\rho[m]]$  then
17      $\rho[m] = i$ 
18 for  $j = (m - 1) \dots 1$  do
19    $\rho[j] = \psi[j + 1][\rho[j + 1]]$ 

```

- *Implicit*. These include temporal expressions such as “Christmas 2001”, “Boxing Day 1995”, or “New Year’s Eve 2000”. Their interpretation requires background knowledge (e.g., that the expression New Year’s Eve implicitly refers to December 31).
- *Relative*. These include temporal expressions such as “yesterday”, “last week”, or “in January”. When interpreting them a temporal anchor (e.g., the publication time of the document) is needed.

Extraction of Temporal Expressions

The extraction of temporal expressions can be separated into an *identification* phase and *interpretation* phase. In the identification phase, parts of the text that constitute temporal expressions are identified. In the interpretation phase, the meaning of the identified temporal expressions is determined by mapping them onto the timeline. For relative temporal expressions this includes determining the right temporal anchor and resolving the temporal expression relative to this anchor. Notice that the separation into these two phases is conceptual – actual tools may interleave the two phases.

State-of-the-art extraction tools for temporal expressions, including as two examples GUTime [MW00] (as part of the TARSQI [VMS⁺05] toolkit) and Timex-Tag [AvRdR07], differ in how much they rely on hand-crafted versus learnt rules. GUTime [MW00], on the one hand, uses a hand-crafted set of regular expressions to identify temporal expressions; for interpreting them a combination of hand-crafted rules and learnt rules is employed. Machine learning is applied, for instance, to distinguish whether “today” refers to the publication time of the document or the meaning of *nowadays*. TimexTag [AvRdR07], on the other hand, relies entirely on statistical learning both in the identification and the interpretation phase.

TimeML

TimeML [PCI⁺03, TIMEML] is a markup language to annotate temporal expressions and event expressions in natural language text. TimeML annotates temporal expressions by enclosing them with the `TIMEX3` tag. Figure 2.3 shows an excerpt from a New York Times article, published on February 1, 2007, with temporal expressions annotated using TARSQI. Let us explain some features of

Hong Kong is poised to hold the first election in more than half **<TIMEX3 tid="t3" TYPE="DURATION" VAL="P100Y">a century</TIMEX3>** that includes a democracy advocate seeking high office in territory controlled by the Chinese government in Beijing. A pro-democracy politician, Alan Leong, announced **<TIMEX3 tid="t4" TYPE="DATE" VAL="20070131">Wednesday</TIMEX3>** that he had obtained enough nominations to appear on the ballot to become the territory's next chief executive. But he acknowledged that he had no chance of beating the Beijing-backed incumbent, Donald Tsang, who is seeking re-election. Under electoral rules imposed by Chinese officials, only 796 people on the election committee -- the bulk of them with close ties to mainland China -- will be allowed to vote in the **<TIMEX3 tid="t5" TYPE="DATE" VAL="20070325">March 25</TIMEX3>** election. It will be the first contested election for chief executive since Britain returned Hong Kong to China in **<TIMEX3 tid="t6" TYPE="DATE" VAL="1997">1997</TIMEX3>**. Mr. Tsang, an able administrator who took office during the early stages of a sharp economic upturn in **<TIMEX3 tid="t7" TYPE="DATE" VAL="2005">2005</TIMEX3>**, is popular with the general public. Polls consistently indicate that three-fifths of Hong Kong's people approve of the job he has been doing. It is of course a foregone conclusion -- Donald Tsang will be elected and will hold office for **<TIMEX3 tid="t9" beginPoint="t0" endPoint="t8" TYPE="DURATION" VAL="P5Y">another five years</TIMEX3>**, said Mr. Leong, the former chairman of the Hong Kong Bar Association.

Figure 2.3: New York Times article annotated using TARSQI

TimeML and TARSQI by means of this example. TimeML distinguishes different types of temporal expressions – in the example the two types `DURATION` and `DATE` are present. Temporal expressions have unique identifiers in TimeML. The publication time of the document –if available– has the unique identifier `t0`. When interpreting a temporal expression, TARSQI takes into account the publication time of the document. The second temporal expression found (i.e., “Wednesday”) is thus correctly interpreted as January 31, 2007, as can be seen from the value `20070131` of the `VAL` attribute. For the last temporal expression found (i.e., “another five years”) TARSQI correctly determines that this refers to a five-year period (as can be seen from the value `P5Y` of the `VAL` attribute). The begin boundary of this duration is determined as the publication time of the document (hence `beginPoint="t0"`); its end boundary refers to the temporal expression having the identifier `t8` that is thus introduced only implicitly.

Our description in this section focuses on temporal expressions and their extraction. For a broader perspective on temporal information extraction and temporal issues in natural language processing, we refer to the survey by Verhagen and Moszkowicz [VM09] and the textbook by Mani et al. [MPG05].

2.3 Temporal Databases

Temporal databases manage data that comes with associated temporal information. This temporal information can refer to different time dimensions:

Transaction time, on the one hand, refers to the time when a fact was stored in the database. In contrast to a regular database that keeps only the currently-valid state of the data, a *transaction-time database* records the history of the data, that is, it keeps track of all states the data was in. Consider as example *transaction-time database* one that manages flight bookings – when passenger Franz Ferdinand books a ticket for flight LH42, conceptually the following record is inserted into the database

```
Franz Ferdinand, LH42, t13, now ,
```

where `t13` is time when the transaction is committed that inserts the record and `now` is a special value that always points to the current time. In the case where our passenger cancels his booking, the record is updated as follows

```
Franz Ferdinand, LH42, t13, t27 ,
```

where t_{27} is the time when the transaction deleting the booking is committed.

Valid time, on the other hand, refers to the time when a fact becomes effective in the real world. Consider as an example *valid-time database* one that manages the history of soccer players' affiliations with teams – an example record in such a database would be

Pelé, New York Cosmos, 1975, 1976

capturing that the famous soccer player Pelé played for the team of New York Cosmos from 1975 until 1976.

One important difference between the two time dimensions, with major implications on the design of index structures, is what is commonly assumed about their evolution. Transaction time is assumed to evolve linearly. Thus, there is no way to change the past. In our above example, there is no way to modify the passenger's booking record, say, to flight LH66. Index structures for transaction-time databases, as a consequence, only have to support modifications of the currently-valid state of the data. For valid time no such assumption is made and arbitrary modifications of the data are possible. In our above example, one could thus insert the record

Pelé, Santos, 1956, 1975,

thus adding information about Pelé's earlier affiliation.

Temporal databases aim at supporting arbitrary data that comes with associated temporal information. This is in contrast to our techniques presented in Chapter 3 that are specifically tailored to text data. Research in temporal databases has looked at various aspects including data models, query languages, and indexing techniques. The latter are closely related to our work and we next describe some of the proposed approaches in brief. Our focus here is on indexing techniques for transaction time. This is because, for the techniques presented in Chapter 3 we also make the assumption that data evolves linearly, so that there are no changes to the past.

2.3.1 Indexing Techniques

Before sketching the ideas underlying some of the indexing techniques proposed for temporal databases, let us introduce some notation and characterize the query types that need to be supported efficiently.

Data Model

The data items (or, records) that we index have the following general form

$$(key, info, t_b, t_e),$$

where *key* is an identifier, *info* is a time-varying piece of information, and $[t_b, t_e]$ is the time interval when the record version was current. As can be seen from our example above, the current version of a record has a special value *now* as its right time-interval boundary t_e . This special value *now* always refers to the current time. The time-interval boundary t_e is updated, once a newer version of the record is added to the database.

Query Types

We focus on four query types that differ in whether they ask for a specific value or range in the key and time dimension, respectively. For the key dimension queries can either ask for records (i) with a specific key *k* or (ii) having a key in a given key range $[k_l, k_h]$. For the time dimension queries can either ask for records that existed (i) at a given time point *t* or (ii) at any time during a given time interval $[t_b, t_e]$. Combining the two dimensions gives us a total of four different query types:

- Key Time-Point Query: $k@t$.
- Key Time-Interval Query: $k@[t_b, t_e]$.
- Key-Range Time-Point Query: $[k_l, k_h]@t$.
- Key-Range Time-Interval Query: $[k_l, k_h]@[t_b, t_e]$.

When describing the different index structures in the following, we discuss which of the four query types are supported by the respective index structure. Notice that the queries have a set-based semantics, i.e., there is no requirement or guarantee regarding the order of the query result.

Multi-Version B-Tree

Several attempts have been made to adapt the B-Tree [Com79] and its variants to deal with time-evolving data. Early approaches like the TSB-Tree [LS89] do

not provide good worst-case guarantees. The Multi-Version B-Tree (MVBT) proposed by Becker et al. [BGO⁺96] solves this problem, as we detail below. Further, when only one version per record is managed, the MVBT achieves the same asymptotic space complexity, as well as the same asymptotic time complexities for key and key-range queries as the B-Tree.

The MVBT, like a B⁺-Tree, keeps records in its leaf nodes and only routing information in its inner nodes. A router stored in an inner node has the form $\langle \text{key}, t_b, t_e \rangle$ where *key* is a separator and $[t_b, t_e]$ is the time interval covered by the subtree rooted at this router. Further, the MVBT tree retains two invariants for each node in the tree. The *weak version condition* demands that for each non-root node and each *t* the number of node entries alive at *t* must either be zero or at least *d*, where $B = k \cdot d$ (i.e., the block size *B* is a multiple of *d*). The weak version condition ensures that, whenever a node is read (and the corresponding disk block is accessed) for a particular time *t*, $O(B)$ required node entries are obtained, which is important with regard to I/O complexity. When the weak version condition is violated or when a node overflows, the the MVBT performs a so-called *version split* that copies all node entries that are still alive to a new node. The *strong version condition* demands that, after a version split, the newly created node must contain between $(1 + \epsilon) \cdot d$ and $(k - \epsilon) \cdot d$ entries. This ensures that at least $\epsilon \cdot d + 1$ insert operations or deletions must be applied to the newly created node, before it needs to be version split. To retain the strong version condition, i.e., to keep the number of entries in a newly created node in the aforementioned range, the MVBT may merge nodes or perform a *key split* that creates a new node and moves all entries having a key larger or equal than a determined separator key. One particularity of the MVBT is that it can have multiple root nodes. Strictly speaking it is thus a directed acyclic graph of tree nodes rather than a tree. Root nodes partition the time dimension into successive intervals – the tree rooted at each of them contains records versions alive at any time in the associated interval.

When processing a key time-point query $k@t$, the root node whose associated time interval contains *t* is identified. Following that, in the subtree rooted at the identified node, the path leading to a leaf node containing a record having key *k* and alive at time *t* is traversed. In analogy, when processing a key-range time-point query $[k_l, k_h]@t$, first the root node responsible for *t* is identified, then the subtree rooted at it is traversed to collect record versions having a key in the desired key range $[k_l, k_h]$ and alive at time *t*. Although not discussed in the

original paper, also query types involving a time interval can be processed using the MVBT. Conceptually, this requires traversing the subtrees rooted at root nodes responsible for any time during the query time-interval $[t_b, t_e]$. When implemented naïvely, this traversal may visit the same tree node multiple (potentially many) times. Van den Bercken and Seeger [BS96] discuss and compare different ways to implement this traversal, so that repeated visits to tree nodes are avoided.

The MVBT manages n record versions using $O(n/B)$ space where B is the block size of the underlying file system. The time complexity for answering a key time-point query is in $O(\log_B n)$; the time complexity for answering a key-range time-point query is in $O(\log_B n + a/B)$ where a is the number of record versions in the result set. When implemented using the best technique proposed in [BS96], key time-interval queries and key-range time-interval queries are supported in $O(\log_B n + a/B + c/B)$ where a is the number of record versions alive at t_e and c is the number of versions created during $[t_b, t_e]$.

Log-Structured History Data Access Method

The log-structured history data access method (LHAM) proposed by Muth et al. [MOPW00] is designed to support high update rates. LHAM partitions the data into successive components C_0, \dots, C_m of (typically geometrically) increasing size based on record timestamps. This partitioning splits the time domain into a set of successive time intervals, each of which is associated with one of the components. The component C_i contains all records that have a timestamp in its associated time interval $[b_i, e_i)$. Components also differ in their expected frequency of access. Therefore, the component C_0 , keeping the most recent data, is held in main memory to support fast updates. Other components, keeping older data, reside on secondary storage or even archive media (e.g., tape).

Initially, all data is added to the component C_0 . When a component C_i becomes full, a so-called rolling merge is triggered. The rolling merge from component C_i to components C_{i+1} migrates all data from C_i to C_{i+1} and updates their associated time intervals. If, as a result, the component C_{i+1} has become full, a merge between C_{i+1} and its successor C_{i+2} is triggered – hence the modifier “rolling”. Figure 2.4 illustrates LHAM’s rolling merge – in the right part of the figure (“Rolling merge II”) the component C_0 becomes full triggering a sequence of two merges. When the component C_m becomes full, the data is either

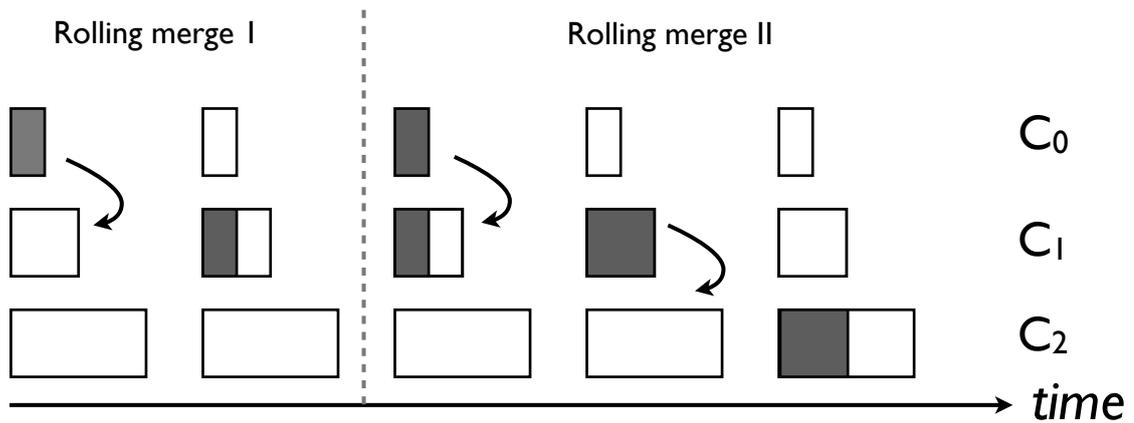


Figure 2.4: LHAM's rolling merge illustrated

purged or a new component C_{m+1} is created.

Within each component data is organized using a secondary index structure – in their implementation the authors use a B^+ -Tree. Records in this secondary index structure bear a compound key consisting of their actual key and their timestamp. Thus, as a concrete example, a data item having key k_{13} and timestamp t_{42} would have the compound key $\langle k_{13} | t_{42} \rangle$ in the secondary index structure.

To process a key time-point query $k@t$, first the component C_i with $t \in [b_i, e_i)$ is identified. Using the secondary index structure the key k is looked up performing a range scan starting from the compound key $\langle k | b_i \rangle$. If no matching record is found, the process continues on the older components C_{i+1}, C_{i+2}, \dots until a matching record has been found. This inspection of older components is required, since a record version still valid at time t could have been created a long time ago and could thus have been migrated to one of the components C_{i+1}, C_{i+2}, \dots . A key time-interval query $k@[t_b, t_e]$ is processed in a similar manner. First, the component C_i with $t_e \in [b_i, e_i)$ is identified. Using the secondary index structure versions with key k created during $[b_i, e_i)$ are identified. Following that, older components C_{i+1}, C_{i+2}, \dots are inspected until a version with a timestamp smaller than t_b has been found. Key-range time-point queries and key-range time-interval queries are processed in analogy to their counterparts just discussed. The sole difference is additional bookkeeping required to keep track of which keys in the given key range have already been completed.

LHAM does not provide non-trivial asymptotic worst-case guarantees. The authors, however, provide a detailed analysis of its average-cost behavior and

demonstrate empirically that it performs superior in practice over the aforementioned TSB-Tree [LS89].

Ramaswamy's Approach

Finally, we discuss an approach proposed by Ramaswamy [Ram97] that somewhat resembles LHAM and, as we detail later, can be seen as a special case of our methods presented in Chapter 3.

Like LHAM Ramaswamy's approach partitions the data along the time dimension, yielding components with successive associated time intervals. One important difference to LHAM is that a component with associated time interval $[b_i, e_i)$ contains all records versions that existed at any point in this time interval. Record versions may thus be redundantly kept in multiple components. Within each component a B⁺-Tree is used as a secondary index structure to manage the contained records. The key idea underlying the approach is to maintain the following invariant for each component: The total number of records contained in a component must not exceed the number of records alive at any time $t \in [b_i, e_i)$ by more than a constant factor $\delta > 1$. Thus, when the component is accessed by any time-point query, at most a fraction $(1 - 1/\delta)$ of the records must be filtered out, because they did not exist at the given time point. This idea of filtering search, i.e., filtering out a small fraction of superfluous data items, was originally proposed by Chazelle [Cha86]. Ramaswamy proposes a greedy online algorithm that maintains this invariant and constructs new components as new record versions are added. Further, it is shown that the space blow-up introduced by the algorithm is at most a constant factor $2\delta/(\delta - 1)$.

A key time-point $k@t$ is processed as follows. First, the component with $t \in [b_i, e_i)$ is identified. Following that, the key k is looked up in the secondary B⁺-Tree. Analogously, a key-range time-point query $[k_l, k_r]@t$ is processed by first identifying the component responsible for t and then performing a range scan on the secondary B⁺-Tree. To support the time-interval query types, the approach has to maintain an additional global B⁺-Tree that organizes record versions using a compound key consisting of their actual key and their left time-interval boundary. A key time-interval query $k@[t_b, t_e]$ is then processed by splitting the query into (i) the key time-point query $k@t_b$ and (ii) a query on the additional global B⁺-Tree that identifies, using a range scan, all record versions having a key k and a left time-interval boundary in (t_b, t_e) . This uses the

idea described by Kannellakis et al. [KRVV96] that all record versions that exist during $[t_b, t_e)$ must either exist at time t_b or have been created during $(t_b, t_e]$. Finally, a key-range time-interval $[k_l, k_h]@ [t_b, t_e]$ is processed by performing a range scan for $[k_l, k_h]$ on the secondary index structure of the component responsible for t_b and one or multiple range scans on the additional global B^+ -Tree that collect all record versions having a key in $[k_l, k_h]$ and a left time-interval boundary in (t_b, t_e) .

For managing n record versions the approach consumes space in $O(n/B)$ where B is the block size of the underlying file system. Further, the approach supports key time-point queries optimally in time $O(\log_B n)$ and key time-interval queries optimally in $O(\log_B n + a/B)$ where a is the number of record versions in the result set. Query types that contain a key range are not supported with optimal asymptotic time complexity.

2.3.2 Temporal Coalescing

Temporal coalescing, as discussed in Böhlen et al. [BSS96], is a unary operator for valid-time databases that groups semantically equivalent records whose valid-time intervals meet. According to Allen's [All83] interval-based temporal logic, two time intervals $[a, b)$ and $[c, d)$ meet if $b = c$. To demonstrate the effect of temporal coalescing, we again employ our above example database that manages soccer players' team affiliations and assume that it contains the following tuples

```
Franz Beckenbauer, Bayern Munich, 1964, 1966
Franz Beckenbauer, Bayern Munich, 1966, 1968
Franz Beckenbauer, Bayern Munich, 1968, 1970.
```

When applying temporal coalescing, the three tuples are grouped as

```
Franz Beckenbauer, Bayern Munich, 1964, 1970,
```

thus capturing that Franz Beckenbauer played for the team of Bayern Munich from 1964 until 1970.

As the example demonstrates, temporal coalescing can significantly reduce the overall number of records in a valid-time database. This positively affects query-processing performance, since database operators have to process fewer tuples. Apart from that, temporal coalescing plays an important role to ensure

the semantics of temporal database operators. To see this, consider a query asking for players who played longer than five years for Bayern Munich. Without coalescing, Franz Beckenbauer is not reported, since none of the affiliations stored in the non-coalesced table lasts five years or longer.

Böhlen et al. [BSS96] show that temporal coalescing can be implemented efficiently both externally (i.e., using only SQL) and internally of an existing database management system (DBMS).

For a more detailed introduction to temporal databases, we refer to Tansel et al. [TCG⁺93]. Jensen et al. [JDB⁺97] provide a comprehensive glossary of terms related to temporal databases. The excellent survey by Salzberg and Tsoutras [ST99] provides an overview and detailed comparison of relevant index structures.

2.4 Web Archiving

The Web evolves rapidly, as studies conducted by Ntoulas et al. [NCO04], Fetterly et al. [FMNW04], and most recently Adar et al. [ATDE09] have shown. On the one hand, fresh web content is constantly added, leading to an overall growth of the Web. On the other hand, web content is removed and disappears forever. Web content is thus ephemeral, and the reasons for its disappearance are manifold. Consider, as two real-world examples, the weekly schedule of a theater that is regularly superseded by newer content and a company website that disappears because the company behind it went out of business. Web archiving counteracts this loss of web content and seeks to preserve it in a durable manner. In the following, we give an overview of web archiving efforts, issues, and current practices.

Preservation Efforts Early in the Web's existence, its ephemeral nature and the need to preserve its contents were recognized. The Internet Archive [IA], as the nowadays arguably best-known endeavor in web archiving, and several national libraries started as early as 1996 to archive parts of the Web. Nowadays, organizations dedicated to the preservation of the Web include non-profit organizations (e.g., the aforementioned Internet Archive [IA] and the European Archive [EA]), public organizations (e.g., national libraries focused on preserv-

Welcome to the



m p i
I N F O R M A T I K

Max-Planck-Institut für Informatik

Im Stadtwald, Gebäude 46.1, D-66123 Saarbrücken, Germany
Phone: +49 681 9325-0, Fax: +49 681 9325-999

- Scientific Working Groups
 - [Algorithms and Complexity Group](#) - AG 1, [Kurt Mehlhorn](#)
 - [Offers and Vacancies](#)
 - [Programming Logics Group](#) - AG 2, [Harald Ganzinger](#)
- [Central Services Groups](#)
- [Publications](#) including [Research Reports](#), [Diploma Theses](#), [Phd Theses](#)
- [Software Projects](#)
- [Library](#)
- [Bibliographic Information](#)
- [Activities](#) including presentations, conferences, workshops, foundations, seminars
- [Miscellaneous](#) including the photo gallery and web server statistics
- [Phone and Room Directory](#)
- [How to reach MPII](#)
- [Intranet](#) (access restricted)

Links to [Max-Planck-Gesellschaft zur Förderung der Wissenschaften e. V.](#)
[Universität des Saarlandes-PhilFak](#) - [FB 14](#) | [DFKI](#) | [IBFI](#)

www page design by Uwe Brahm <webmaster@mpi-sb.mpg.de>, 20 february 1997

Figure 2.5: URL <http://www.mpi-sb.mpg.de> as of March 5, 1997 archived by the Internet Archive (<http://www.archive.org>)

ing their local part of the Web), and private companies (e.g., archiving their own or a client’s web site). Figure 2.4, as a concrete example, shows a snapshot of the URL <http://www.mpi-sb.mpg.de> from March 5, 1997 that can be found in the Internet Archive. Equally diverse are the motivations behind preservation efforts that range from preserving a part of our cultural heritage for future generations (e.g., for national libraries) to having a record of all published contents as legal evidence (e.g., for a company archiving its own website). With the International Internet Preservation Consortium (IIPC) [IIPC] several web-preserving organizations have established a body that, among other things, seeks to establish common standards and develop common software. Recently, the idea of establishing a decentralized web archive that puts the load on the shoulders of many peers has been studied; a conceivable architecture is discussed in Anand et al. [ABB⁺09a].

Issues & Current Practices Every web preservation effort must address two important issues. The first issue concerns the *scope* of the effort, i.e., which parts of the Web it seeks to preserve. For national libraries, as said above, the scope would typically be *domain-centric* including all websites hosted under a specific top-level domain (e.g., *.de*, *.uk*, or *.cn*). Scopes can also be *topic-centric* including all websites related to a certain topic or event (e.g., the 2008 U.S. presidential election). When only a single website is archived (e.g., for legal purposes) the scope is *site-centric*. The second issue concerns the *acquisition*, i.e., how web content is actually collected for preservation. One mode of acquisition is the so-called *transaction archiving* in which the web server is equipped to record in detail every request received and every response sent. An alternative mode, also operating at the server side, is *server-side archiving* that periodically copies files hosted by the web server to an archive. Finally and most commonly, web content is collected at the client side using so-called *client-side archiving*. To this end, like in commercial web search engines, a web crawler is employed that requests and stores web content by systematically following references (e.g., hyperlinks) between files. Crawling for web preservation, though, differs in two important aspects from the crawling that web search engines perform. First, since a complete preservation is desired so that archived web content can be displayed as it was, the crawler needs to fetch all files and not only the ones that will be indexed for search. Second, to avoid an overload of the web server hosting the content to be archived, the crawler must be polite and pause for a couple of seconds between requests. As a result of this politeness, crawling a web site may take up to several weeks. This clearly impedes preserving an accurate and complete copy of the website, since the website may undergo massive changes while crawling takes place. The open-source crawler Heritrix [HERITRIX] jointly developed by members of the IIPC is such a crawler specifically designed for web preservation.

For a more comprehensive account on issues and current practices in web archiving we refer to Masanès [Mas06]. Living Web Archive (LiWA) [LIWA], as a second reference, is an ongoing research project funded by the European Commission that addresses open issues in web archiving, among them the issue of terminology evolution that we consider in Chapter 4.

Chapter 3

Efficient

Time-Travel Text Search in Web Archives

3.1 Motivation & Problem Statement

In this chapter, we address *time-travel text search* in web archives and other versioned document collections. Given a query q and the user's time of interest, specified as a time point t or a time interval $[t_b, t_e]$, our goal is to *efficiently identify and rank document versions that match the user's query and existed at the specified time of interest*.

An increasing number of versioned document collections is available today, including web archives, collaborative authoring environments like Wikis (with Wikipedia [WIKI] as the prime example), or timestamped information feeds. Text search on such versioned document collections, however, is mostly time-ignorant: while the searched document collection changes over time, often only the most recent version of a document is indexed and searched or, alternatively, versions are indexed independently and treated as separate documents. Even worse, for some collections, particularly web archives like the Internet Archive [IA], a comprehensive text-search functionality, as provided by commercial search engines like Google, Yahoo!, and Bing for the current Web, is often missing completely.

Time-travel text search, as we develop it in this chapter, is a crucial tool to search and explore versioned document collections and thus to unfold their full potential, as the following two use cases demonstrate:

- For a documentary about a political scandal years ago, a journalist needs to research early opinions and statements made by the involved politicians. Sending an appropriate query to a major web search-engine, the majority of returned results contains only recent coverage, since many of the early web pages have disappeared and are only preserved in web archives. If the query could be enriched with a time point of interest, say *August 20th 2003* as the day after the scandal got revealed, and be issued against a web archive, only pages that existed specifically at that time could be retrieved thus better satisfying the journalist's information need.
- A business analyst wants to identify leading figures in the technology world. To this end, our analyst focuses on recently established technologies and tries to identify early proponents and adopters. However, issuing relevant queries such as *ajax* or *solid-state disks* against a commercial search engine, our analyst sees only recent documents published after the technologies became established. Again, if the queries could be enriched with a time interval of interest, say *January 2005 through March 2006*, and be issued against a web archive, our analyst could identify leading figures by looking at early documents discussing the techniques.

Document collections like the Web or Wikipedia [WIKI], as we target them here, are already large if only a single snapshot is considered. Looking at their evolutionary history, we are faced with even larger volumes of data.

Architectural Alternatives

To implement the desired time-travel text search functionality, one may resort to the traditional inverted file index [ZM06], which is used by commercial web-search engines and thus known to cope with the scale of the Web. The inverted file index, however, is time-ignorant – it does not provide efficient means to retrieve only information relevant to the user's time of interest. Clearly, one may filter-out information that is not relevant when processing a query, the amount of data read, however, then depends on the size of the document collection, but not on the user's time of interest. Another conceivable approach to implement time-travel text search is to use indexing techniques originally proposed for temporal databases such as the Multi-Version B-Tree [BGO⁺96] or LHAM [MOPW00], which we described in detail in Chapter 2. Although

these techniques provide means to efficiently retrieve information relevant to the user’s time of interest, their applicability to text search is not well-understood. Furthermore, it is unclear whether they can work smoothly together with other techniques that are key to efficient text search as, for instance, query processing techniques that depend on postings being retrieved in a particular order, as well as compression and encoding techniques.

The naïve application of existing techniques thus fails to provide a viable solution to time-travel text search. We therefore follow a different approach in this chapter and propose an efficient approach to time-travel text search that builds on the inverted file index but integrates ideas originally proposed for temporal databases.

Contributions

In detail, we make the following key contributions in this chapter:

- The *Time-Travel Inverted indeX* (TTIX) is proposed as a versatile framework to index and search versioned document collections that extends the popular well-studied inverted index [ZM06].
- *Temporal coalescing techniques* are introduced that target different types of posting payloads and avoid an index-size explosion at index-build time while keeping query results highly accurate.
- We develop *partitioning strategies* that allow trading off index size and query-processing performance. Using these strategies the time-travel inverted index can be fine-tuned at index-build time according to imposed query-processing performance requirements or space constraints.
- In a *comprehensive experimental evaluation* our approach is evaluated on the revision history of the English Wikipedia, parts of the European Archive, and the New York Times Annotated Corpus as three representative large-scale real-world versioned document collections.

Organization

The remainder of this chapter is organized as follows. We put our work in context with related research in Section 3.2. Subsequently, we delineate our model of a versioned document collection, our query model, and adapt the Okapi BM25

retrieval model for time-travel text search. The time-travel inverted index, as the central building block in our approach, is introduced in Section 3.4. The processing of time-point queries and time-interval queries on the time-travel inverted index is the subject of Section 3.5. Section 3.6 presents temporal coalescing techniques for different payload types that keep the index compact. Partitioning strategies to fine-tune the time-travel inverted index are described in Section 3.7. In Section 3.8, we describe how time-dependent collection statistics, as needed by our retrieval model, can be managed efficiently. Our implementation of the proposed techniques in a prototype system called FLUXCAPACITOR is described in Section 3.9. Following that, Section 3.10 provides details on our experimental evaluation of the proposed techniques and its results. We conclude this chapter in Section 3.11 and point out future directions of research.

3.2 Related Work

We can classify the related work mainly into the following three categories: (i) methods that deal explicitly with search on versioned document collections, (ii) methods for reducing the index size by exploiting either overlap between document contents or by pruning portions of the index, and (iii) indexing techniques originally proposed in the context of temporal databases. We now review work from each of these three categories.

Search on Versioned Document Collections

There is only little prior work dealing with search in versioned document collections. Anick and Flynn [AF92], who pioneered this research, describe a helpdesk system that supports historical queries. Access costs are optimized for accesses to the most recent versions and increase as one moves farther into the past. Burrows and Hisgen [BH99], in a patent description, delineate a method for indexing range-based values and mention its potential use for searching based on publication dates associated with documents. Using the same argument, the more recent approach proposed by Fontoura et al. [FLQZ07] that extends the inverted index to support search on numerical domains (e.g., product prices) could also be leveraged to search based on dates. Note that searching merely based on documents' publication dates is different from the time-travel text search that we develop in this work – time-travel text search looks at when

documents existed as opposed to when they were published. Recent work by Nørvåg and Nybø [NN06] and their earlier proposals [Nør03, Nør04] concentrate on the problem of supporting Boolean queries and neglect the relevance scoring of results. He et al. [HYS09], most recently, examine different encoding schemes and their effectiveness when indexing a versioned document collection. Stack [Sta06] reports practical experiences made when adapting the open source search-engine Nutch to search web archives. This adaptation, however, does not provide the intended time-travel text search functionality. Finally, although not focused on search, we mention the Cornell Web Library [AAD⁺06] and Zoetrope [ADFW08] as two systems that provide tools to interact with and mine web archives.

Reduction of Index Size

Moving on to the second category of related work, Broder et al. [BEF⁺06] describe a technique that exploits large content overlaps between documents to achieve a reduction in index size. Their technique makes strong assumptions about the structure of document overlaps, which can be observed for special document collections such as e-mail conversations, but are too restrictive in our context, thus rendering the technique inapplicable. More recent approaches by Herscovici et al. [HLY07] and Zhang and Suel [ZS07] exploit arbitrary content overlaps between documents to reduce index size. None of the approaches, however, considers time explicitly or provide the desired time-travel text search functionality. Static index-pruning techniques [AM06b, CCF⁺01, NC07] aim to reduce the effective index size, by removing portions of the index that are expected to have low impact on the query result. They also do not consider temporal aspects of documents, and thus are technically quite different from our proposal despite having a shared goal of index-size reduction. It should be noted, though, that index-pruning techniques can be adapted to work along with our time-travel inverted index.

Indexing Techniques for Temporal Databases

Research in temporal databases, as discussed in Chapter 2, has produced a number of index structures tailored for time-evolving data. It is conceivable to use index structures like the Multi-Version B-Tree [BGO⁺96] or LHAM [MOPW00] for an implementation of time-travel text search as we describe it here. Their

applicability and practical performance for text search, though, are not well-understood. Further, it is unclear whether existing techniques that are considered key to efficient text search (e.g., the compression and pruning techniques discussed in Chapter 2) can easily be used in conjunction with these approaches. Lomet et al. [LHNZ08] is one recent approach that integrates a compression technique similar to temporal coalescing into the MSB-Tree as an index structure for time-evolving data.

3.3 Model

We next introduce our formal model and the notation that will be used throughout the rest of this chapter.

3.3.1 Time Domain & Collection Model

We deal with a versioned document collection \mathbf{D} . Each document $\mathbf{d} \in \mathbf{D}$ is a sequence of its versions

$$\mathbf{d} = \langle d^{t_1}, d^{t_2}, \dots \rangle, \quad (3.1)$$

ergo we assume a linear evolution of the document. Each version d^{t_i} has an associated timestamp t_i reflecting when the version was created. Each version is a bag of searchable terms drawn from a vocabulary \mathcal{V} . Any modification to a document version results in the insertion of a new version with a corresponding timestamp. We employ a discrete definition of time and assume the integers \mathbb{Z} as our time domain \mathcal{T} with timestamps $t \in \mathcal{T}$ denoting the number of time units (e.g., milliseconds or days) passed (to pass) since (until) a reference time-point (e.g., the UNIX epoch). These time units will be further referred to as chronons. The deletion of a document at time t_i , i.e., its disappearance from the current state of the collection, is modeled as the insertion of a special “tombstone” version \perp . The valid-time interval $\text{val}(d^{t_i})$ of a version d^{t_i} is $[t_i, t_{i+1})$, if a newer version with associated timestamp t_{i+1} exists, and $[t_i, \text{now})$ otherwise where now points to the current time and is assumed to be larger than any known timestamp.

Building on this, we define the collection at a time point t and, as a generalization, the collection during a time interval $[t_b, t_e]$ as captured in the following two definitions.

Definition 3.1 (Document collection at time point) *The state \mathbf{D}^t of the collection at time t (i.e., the set of versions valid at t that are no deletions) is*

$$\mathbf{D}^t = \bigcup_{d \in \mathbf{D}} \{d^{t_i} \in \mathbf{d} \mid t \in \text{val}(d^{t_i}) \wedge d^{t_i} \neq \perp\}. \quad (3.2)$$

Definition 3.2 (Document collection during time interval) *The state $\mathbf{D}^{[t_b, t_e]}$ of the collection during the time interval $[t_b, t_e]$ (i.e., the set of all versions that were valid at any time $t \in [t_b, t_e]$ and are no deletions) is*

$$\mathbf{D}^{[t_b, t_e]} = \bigcup_{d \in \mathbf{D}} \{d^{t_i} \in \mathbf{d} \mid [t_b, t_e] \cap \text{val}(d^{t_i}) \neq \emptyset \wedge d^{t_i} \neq \perp\}. \quad (3.3)$$

Note that, by definition, \mathbf{D}^t contains at most one version per document. In contrast, more than one version of a document can be contained in $\mathbf{D}^{[t_b, t_e]}$.

3.3.2 Query Model

As mentioned earlier, our objective is to enrich text search with time-travel functionality. Let q be a query and t be a timestamp. Note that q can be a Boolean query, keyword query, or phrase query as introduced in Chapter 2. We employ q^t to refer to the *time-point query* that evaluates q on \mathbf{D}^t , thus taking into account all document versions alive at time t . In analogy, we use $q^{[t_b, t_e]}$ to refer to the *time-interval query* that evaluates q on $\mathbf{D}^{[t_b, t_e]}$, then considering all document versions alive at any time in $[t_b, t_e]$.

3.3.3 Retrieval Model

In this work, as a retrieval model for keyword queries, we adopt Okapi BM25, as described in Chapter 2, but note that the proposed techniques are not dependent on this choice and are applicable to other retrieval models as well. For our considered setting, we slightly adapt Okapi BM25 to make it time-aware.

Definition 3.3 (Time-aware Okapi BM25) *Let $q^{[t_b, t_e]}$ be a time-travel keyword query and d^{t_i} be a document version, the document version's relevance to the query is*

$$w(q^{[t_b, t_e]}, d^{t_i}) = \sum_{v \in q} w_{\text{tf}}(v, d^{t_i}) \cdot w_{\text{idf}}(v, [t_b, t_e]). \quad (3.4)$$

We reiterate that $q^{[t_b, t_e]}$ is evaluated over $\mathbf{D}^{[t_b, t_e]}$ so that only versions d^{t_i} alive at any time in $[t_b, t_e]$ are considered.

The tf-score $w_{\text{tf}}(v, d^{t_i})$ is defined as

$$w_{\text{tf}}(v, d^{t_i}) = \frac{(k_1 + 1) \cdot \text{tf}(v, d^{t_i})}{k_1 \cdot ((1 - b) + b \cdot \frac{\text{dl}(d^{t_i})}{\text{avdl}}) + \text{tf}(v, d^{t_i})}, \quad (3.5)$$

considering the plain term frequency $\text{tf}(v, d^{t_i})$ of term v in version d^{t_i} normalizing it, taking into account both the length $\text{dl}(d^{t_i})$ of the version and the average document length avdl in the collection. Note that we assume the average document length avdl to be stationary – a reasonable assumption for most versioned document collections. The length-normalization parameter b and the tf-saturation parameter k_1 are inherited from the original Okapi BM25 and are commonly set to values 0.75 and 1.2, respectively.

The idf-score $w_{\text{idf}}(v, [t_b, t_e])$ reflects the inverse document frequency of term v in document versions contained in $\mathbf{D}^{[t_b, t_e]}$ and is defined as

$$w_{\text{idf}}(v, [t_b, t_e]) = \log \frac{N([t_b, t_e]) - \text{df}(v, [t_b, t_e]) + 0.5}{\text{df}(v, [t_b, t_e]) + 0.5}, \quad (3.6)$$

where $N([t_b, t_e]) = |\mathbf{D}^{[t_b, t_e]}|$ is the number of document versions alive at any time in $[t_b, t_e]$ and $\text{df}(v, [t_b, t_e])$ gives the number of those versions that contain the term v .

While the idf-score in the above definition depends on the whole document collection and on the time interval $[t_b, t_e]$, the tf-score is specific to each version and can thus be precomputed. Techniques to compute time-dependent idf-scores efficiently at query-processing time are described in Section 3.8.

3.4 Time-Travel Inverted Index

The inverted index, discussed in Chapter 2, is the standard technique for text indexing deployed in many of today’s systems. In this section, we present the Time-Travel Inverted index (TTIX) as a versatile framework that extends the inverted index and makes it ready for time-travel text search.

In order to support time-travel text search, TTIX extends both the structure of postings and the lexicon by explicitly incorporating temporal information. Figure 3.1 shows a TTIX instance with postings having scalar payloads that capture plain term frequencies.

Posting Structure

Postings in TTIX are extended by including a *valid-time interval* $[t_i, t_j)$ to denote when the payload information was valid in the real world. In detail, postings in

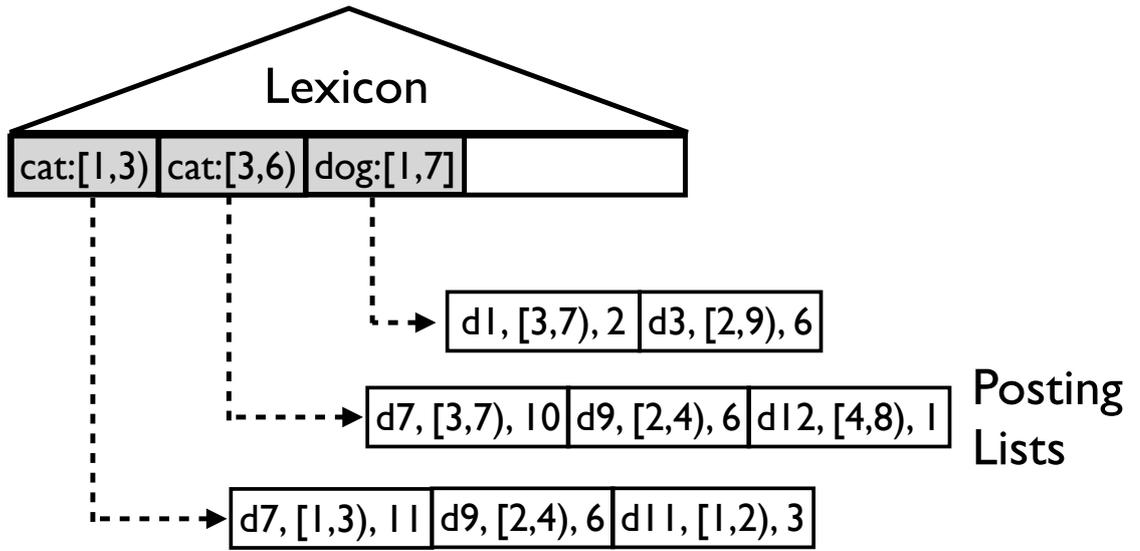


Figure 3.1: Time-Travel Inverted index (TTIX) example instance

TTIX have the form

$$(d, [t_i, t_j), p),$$

where d and p are a document identifier and a payload, as in the standard inverted index, and $[t_i, t_j)$ is a valid-time interval. The posting

$$(d_1, [3, 7), 2)$$

in Figure 3.1, as an example of a posting with a scalar payload, conveys that versions of document d_7 alive in $[3, 7)$ contain two occurrences of the term *dog*.

Temporal Partitioning

In contrast to the standard inverted index that maintains one posting list L_v per term v from the lexicon, TTIX supports temporal partitioning and potentially maintains multiple posting lists for each term. Temporal partitioning is key to optimizing query-processing performance. Posting lists in TTIX have an associated time interval $[t_k, t_l)$, and the posting list $L_v : [t_k, t_l)$ contains all postings for the term v with a valid-time interval that overlaps with $[t_k, t_l)$, i.e.,

$$L_v : [t_k, t_l) = \{(d, p, [t_i, t_j)) \in L_v \mid t_i < t_l \wedge t_j > t_k\}, \quad (3.7)$$

where L_v refers to the list of all postings belonging to term v . Note that the list $L_v : [t_k, t_l)$ alone can be used to retrieve all postings for the term v and any

query time interval $[t_b, t_e] \subseteq [t_k, t_l]$. In the following, we refer to the *temporal partitioning for term v* as \mathcal{P}_v , i.e., the set of time intervals associated with posting lists for the term v in our index. We demand that posting lists with an associated time interval in \mathcal{P}_v cover L_v completely, i.e., every posting contained in L_v is contained in at least one of the posting lists that we keep in our index. Formally, this requirement can be stated as

$$L_v = \bigcup_{[t_k, t_l] \in \mathcal{P}_v} L_v : [t_k, t_l]. \quad (3.8)$$

Later, in Section 3.7, we describe partitioning strategies to systematically determine \mathcal{P}_v and thus to decide which posting lists should be kept in our index for term v .

Figure 3.1 shows an example of a TTIIX instance. For the term `cat`, the index maintains two posting lists `cat:[1, 3)` and `cat:[3, 6)` covering the time intervals $[1, 3)$ and $[3, 6)$, respectively. Further, note that the posting $(d_9, [2, 4), 6)$ is replicated and contained in both posting lists, since its valid-time interval overlaps with the time intervals of both posting lists.

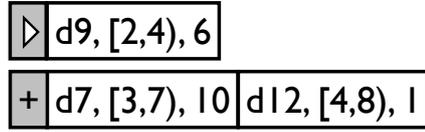


Figure 3.2: Physical storage of posting list `cat:[3, 6)` from Figure 3.1

To speed up query processing for time-interval queries, as we explain in Section 3.5, we store the *logical* posting list $L_v : [t_k, t_l]$ in two separate *physical* posting lists on disk, namely

$$L_v^< : [t_k, t_l) = \{(d, p, [t_i, t_j)) \in L_v \mid t_k \in (t_i, t_j)\}, \quad (3.9)$$

as the list of replicated postings already alive before time t_k , and

$$L_v^+ : [t_k, t_l) = \{(d, p, [t_i, t_j)) \in L_v \mid t_i \in [t_k, t_l)\}, \quad (3.10)$$

as the posting list containing all postings created during $[t_k, t_l)$. Note that the two posting lists are disjoint, i.e.,

$$L_v^< : [t_k, t_l) \cap L_v^+ : [t_k, t_l) = \emptyset \quad (3.11)$$

and contain all postings from $L_v : [t_k, t_l)$, i.e.,

$$L_v^{\triangleright} : [t_k, t_l) \cup L_v^{\triangleleft} : [t_k, t_l) = L_v : [t_k, t_l) . \quad (3.12)$$

Figure 3.2 shows the two physical posting lists that we store on disk to represent the logical posting list $\text{cat}:[3, 6)$ from Figure 3.1.

Posting-List Order

TTIX does not restrict the internal sort order of posting lists, but assumes that it is consistent for all posting lists in the index. Posting lists can be, for instance, document-ordered, score-ordered, or ordered taking into account the postings' valid-time interval boundaries. By not restricting the order of posting lists, query-processing techniques and other optimizations proposed for the standard inverted index (e.g., early-terminating query-processing techniques that rely on score-ordered posting lists or compression techniques that leverage ordered document identifiers) remain equally applicable for TTIX.

3.5 Query Processing

We now describe how time-travel queries are processed in TTIX, i.e., how postings relevant for a given time-travel query can be retrieved efficiently.

3.5.1 Time-Point Queries

Given a time-point query q^t , for each query term $v \in q$ we must retrieve all postings for the term v whose valid-time interval $[t_i, t_j)$ contains the query time-point, i.e., $t \in [t_i, t_j)$. This can be accomplished by selecting an appropriate posting list $L_v : [t_k, t_l)$ from our index that fulfills $t \in [t_k, t_l)$. Obviously, we would like to reduce the total number of postings read from the index. Therefore, if there are multiple qualifying posting lists, we choose the shortest one. We thus solve the following optimization problem for each term v :

Definition 3.4 (Time-point query optimization problem)

$$\operatorname{argmin}_{[t_k, t_l) \in \mathcal{P}_v} |L_v : [t_k, t_l)| \quad \text{s.t.} \quad t \in [t_k, t_l) ,$$

where $|L_v : [t_k, t_l]|$ denotes the length of posting list $L_v : [t_k, t_l]$ and is defined as

$$|L_v : [t_k, t_l]| = |L_v^> : [t_k, t_l]| + |L_v^+ : [t_k, t_l]| . \quad (3.13)$$

Following that, the two posting lists $L_v^> : [t_k, t_l]$ and $L_v^+ : [t_k, t_l]$ are read and merged, thereby exploiting their common sort order. While doing so, irrelevant postings with a valid-time interval $[t_i, t_j)$ such that $t \notin [t_i, t_j)$ are filtered out.

Whether a posting is filtered out can only be decided after the posting has been transferred from disk into main memory and therefore still incurs significant I/O cost. Query-processing performance for time-point queries therefore depends heavily on the amount of read but filtered-out postings. The partitioning strategies presented in Section 3.7 determine partitionings at index-build time to reduce the I/O overhead incurred by filtered-out postings significantly.

3.5.2 Time-Interval Queries

Given a time-interval query $q^{[t_b, t_e]}$, we must retrieve for each query term $v \in q$ all postings from the index for the term v whose valid-time interval overlaps with the query time-interval, i.e., $[t_i, t_j) \cap [t_b, t_e] \neq \emptyset$.

Note that, in contrast to the processing of time-point queries described above, there may not be a single list $L_v : [t_k, t_l)$ such that $[t_b, t_e] \subseteq [t_k, t_l)$ which could thus alone be used to answer the query. Recall that \mathcal{P}_v denotes the set of time intervals for which our index keeps a posting list for term v . To find all postings relevant to term v and time interval $[t_b, t_e]$, we have to determine a set $\mathcal{L}_v \subseteq \mathcal{P}_v$ of time intervals that together cover the query time-interval $[t_b, t_e]$. Note that we can safely assume that every time interval from \mathcal{L}_v overlaps with the query time-interval $[t_b, t_e]$, i.e.,

$$\forall [t_k, t_l) \in \mathcal{L}_v : [t_k, t_l) \cap [t_b, t_e] \neq \emptyset , \quad (3.14)$$

since we could otherwise remove $[t_k, t_l)$ and still process the query. Similarly, we can assume that there is no subsumption between time intervals in \mathcal{L}_v , i.e.,

$$\forall [t_i, t_j) \in \mathcal{L}_v \forall [t_k, t_l) \in \mathcal{L}_v : [t_i, t_j) \subseteq [t_k, t_l) \Rightarrow [t_i, t_j) = [t_k, t_l) . \quad (3.15)$$

Putting these two together, we can assume that \mathcal{L}_v is a sequence

$$\mathcal{L}_v = \langle [t_{b_1}, t_{e_1}), \dots, [t_{b_m}, t_{e_m}) \rangle \quad (3.16)$$

of m time intervals arranged in ascending order of their begin boundary retaining $t_{e_i} \geq t_{b_{i+1}}$, $t_b \in [t_{b_1}, t_{e_1})$, and $t_e \in [t_{b_m}, t_{e_m})$. To identify relevant postings for the query, we merge the posting lists $L_v^\Delta : [t_{b_1}, t_{e_1})$ and $L_v^+ : [t_{b_i}, t_{e_i})$ for $1 \leq i \leq m$. In doing so, we are guaranteed to read all postings whose valid-time interval overlaps with $[t_{b_1}, t_{e_m})$. This is because of the dichotomy that postings whose valid-time interval overlaps with $[t_{b_1}, t_{e_m})$ are either already alive before time t_{b_1} , which we obtain by reading the posting list $L_v^\Delta : [t_{b_1}, t_{e_1})$, or have been created during (t_{b_1}, t_{e_m}) , which we obtain by reading the posting lists $L_v^+ : [t_{b_i}, t_{e_i})$. Moreover, since $[t_b, t_e] \subseteq [t_{b_1}, t_{e_m})$, we are guaranteed to see all postings relevant to the query. While merging the posting lists, we filter out postings that have a valid-time interval $[t_i, t_j)$ such that $[t_i, t_j) \cap [t_b, t_e] = \emptyset$. The merging and filtering can be implemented efficiently using a priority queue, thus exploiting the fact that posting lists have a consistent sort order. This order is naturally preserved for the merged list of relevant postings.

As for time-point queries, the query-processing performance of a time-interval query depends on the total number of postings read, thus taking into account filtered-out and duplicate postings. For a term v contained in a given time-interval query, the question which sequence \mathcal{L}_v to pick and thus which posting lists to merge can be formalized as the following optimization problem:

Definition 3.5 (Time-interval query optimization problem)

$$\begin{aligned} \operatorname{argmin}_{\mathcal{L}_v} & |L_v^\Delta : [t_{b_1}, t_{e_1})| + \sum_{[t_{b_i}, t_{e_i}) \in \mathcal{L}_v} |L_v^+ : [t_{b_i}, t_{e_i})| \quad s.t. \\ & [t_b, t_e] \subseteq \bigcup_{[t_{b_i}, t_{e_i}) \in \mathcal{L}_v} [t_{b_i}, t_{e_i}). \end{aligned}$$

When choosing \mathcal{L}_v , we thus aim at minimizing the total number of postings read, while making sure that we see all postings relevant for the term v and the time interval $[t_b, t_e]$. The greedy Algorithm 2 computes an optimal solution to the above optimization problem. The algorithm keeps track of an optimal solution l for the time interval $[t_b, t)$ and its associated cost c as a triple (t, c, l) in the set S . In each iteration of the main loop, one additional triple is added to S , by determining the globally cost-minimal solution for a time interval $[t_b, t')$ that has not yet been covered. To this end, the algorithm extends an optimal solution (t, c, l) already recorded in S by appending a time interval $[t_k, t_l)$ from \mathcal{P}_v that does not introduce a gap, i.e., $t \in [t_k, t_l)$. If $[t_k, t_l)$ is the first time interval,

the solution obtained has cost $|L_v^{\triangleright} : [t_k, t_l]| + |L_v^{\triangleleft} : [t_k, t_l]|$. Otherwise, its cost is yielded by incrementing the cost of the extended solution by $|L_v^{\triangleleft} : [t_k, t_l]|$. The algorithm terminates and outputs an optimal solution for $[t_b, t_e]$, once a solution for $t' > t_e$ has been determined.

The algorithm closely resembles Dijkstra's algorithm [KT05] for computing shortest paths in a graph. Note that the pseudo-code in Algorithm 2 is meant to illustrate the ideas underlying the algorithm, but does not achieve good time and space complexities. In detail, if implemented as described, it achieves time complexity in $O(|\mathcal{P}_v|^3)$ and space complexity in $O(|\mathcal{P}_v|^2)$. If a priority queue is used to implement the set S and optimal solutions are represented implicitly using pointers, these can be reduced to $O(|\mathcal{P}_v| \cdot \log |\mathcal{P}_v|)$ time and $O(|\mathcal{P}_v|)$ space, respectively, as discussed for Dijkstra's algorithm in Kleinberg and Tardos [KT05]. The optimality of the algorithm is stated in the following theorem.

Theorem 3.1 *Algorithm 2 determines an optimal sequence \mathcal{L}_v of time intervals.*

We need two lemmas to prove Theorem 3.1.

Lemma 3.1 *Algorithm 2 adds triples in non-decreasing order of their cost to the set S .*

Proof of Lemma 3.1 *By contradiction. Let (t, c, \mathcal{l}) and (t', c', \mathcal{l}') be two triples that the algorithm adds in consecutive iterations to the set S . We assume $c' < c$, i.e., the triple added last has lower cost. This is impossible, since in each iteration Algorithm 2 extends a solution that is already in S . Therefore, either $\mathcal{l} \subset \mathcal{l}'$, i.e., in the last iteration the solution added just before is extended, which implies $c \leq c'$. Or, \mathcal{l}' extends another solution that was already in S when \mathcal{l} was determined, which also implies $c \leq c'$, since otherwise our greedy algorithm would have selected (t', c', \mathcal{l}') in the first iteration. \square*

Lemma 3.2 *For any $(t, c, \mathcal{l}) \in S$ the sequence \mathcal{l} is an optimal solution for $[t_b, t)$.*

Proof of Lemma 3.2 *By induction over $|S|$.*

$|S| = 1$: *For the initial case $S = \{(t_b, 0, \emptyset)\}$ the lemma holds, since no work and therefore zero cost is needed to cover the empty interval $[t_b, t_b)$.*

$|S| = 2$: *Let (t, c, \mathcal{l}) be the triple that Algorithm 2 adds to S in the first iteration of the main loop. By design, in this first iteration, the algorithm chooses the $[t_k, t_l) \in \mathcal{P}_v$ that has minimal cost $|L_v : [t_k, t_l)|$ and fulfills $t_b \in [t_k, t_l)$. Thus, any other solution for $[t_b, t)$ must have a cost of at least c , since otherwise the algorithm would have selected it.*

Algorithm 2: Determining an optimal sequence \mathcal{L}_v of time intervals whose posting lists are merged to retrieve all postings relevant to query term v and query time-interval $[t_b, t_e]$

Data: Set of time intervals \mathcal{P}_v , query time-interval $[t_b, t_e]$

Result: Sequence of time intervals $\mathcal{L}_v \subseteq \mathcal{P}_v$

```

1 /* Initialization */
2  $\mathcal{L}_v = \emptyset$ 
3  $S = \{(t_b, 0, \emptyset)\}$ 
4  $t' = t_b$ 
5 /* Greedy computation */
6 while  $t' \leq t_e$  do
7    $c' = \infty$ 
8    $l' = \emptyset$ 
9   for  $(t, c, l) \in S$  do
10    for  $[t_k, t_l] \in \mathcal{P}_v$  do
11     if  $t \in [t_k, t_l] \wedge \neg \exists (t_l, *, *) \in S$  then
12       $c'' = c$ 
13      if  $l = \emptyset$  then
14        $c'' = c'' + |L_v^> : [t_k, t_l]|$ 
15        $c'' = c'' + |L_v^+ : [t_k, t_l]|$ 
16       if  $c'' < c'$  then
17         $c' = c''$ 
18         $l' = l \cup \{[t_k, t_l]\}$ 
19         $t' = t_l$ 
20    $S = S \cup \{(t', c', l')\}$ 
21    $\mathcal{L}_v = l'$ 

```

$|S| \rightarrow |S| + 1$ for $|S| \geq 2$: Let (t, c, l) be the triple that Algorithm 2 adds to S and let (t', c', l') be the triple that the algorithm extends to this end. Assume that there is a solution (t, c^*, l^*) that achieves cost $c^* < c$. Note that there must be an optimal $(t'', c'', l'') \in S$ such that $l'' \subset l^*$ – this holds, for instance, for the initial triple $(t_b, 0, \emptyset)$ added to S . Therefore, Algorithm 2 could gradually extend (t'', c'', l'') and yield (t, c^*, l^*) . The fact that it extends (t', c', l') into (t, c, l) , when growing S , indicates that any extension of (t'', c'', l'') has cost of at least c . Therefore, (t, c^*, l^*) must have cost $c^* \geq c$, which contradicts our assumption. \square

Proof of Theorem 3.1 Due to Lemma 3.2 we know that the solution l' output by Algorithm 2 is an optimal solution for $[t_b, t')$ that also covers $[t_b, t_e]$ since $t' > t_e$. Any other solution covering $[t_b, t_e]$ must have a cost of at least c' due to Lemma 3.1. \square

As a side remark, note that for the special case $t_b = t_e$ corresponding to a time-point query, Algorithm 2 correctly picks a list $L_v : [t_k, t_l)$ that has shortest length while retaining $t_b \in [t_k, t_l)$, which matches our above description of how time-point queries are processed using TTIX.

Note that the result of a time-interval query may contain more than one version per document. This is in contrast to time-point queries whose results contain at most one version per document. This subtle difference has ramifications on the bookkeeping required during query processing (for time-point queries results can be uniquely identified based on their document identifier) and on the temporal coalescing techniques presented in the following section.

3.6 Temporal Coalescing

If we employ TTIX, as described thus far, to index a versioned document collection, we would naïvely create one posting per term per document version. For frequent terms and highly-dynamic document collections, this leads to a huge number of postings that have to be kept in our index. Often, though, changes to documents are minor (e.g., spelling corrections), leading to a typically high degree of redundancy between consecutive versions of the same document. In this section, we describe techniques that leverage this high degree of redundancy to reduce the total number of distinct postings kept in our index. Toward this

objective, the techniques presented coalesce postings belonging to consecutive (i.e., temporally adjacent) versions of the same document at index-build time and thus construct postings that contain information about consecutive versions of the same document. The difference between the techniques lies in the type of posting payload that they target.

When presenting the temporal coalescing techniques for Boolean, scalar, and positional payloads in the following, we assume that our input consists of a sequence of n temporally adjacent postings

$$I = \langle (d, [t_1, t_2], p_1), \dots, (d, [t_n, t_{n+1}], p_n) \rangle. \quad (3.17)$$

The input thus represents a contiguous time period during which the term was present in the document d . If the term disappears from d but reappears later, multiple input sequences are obtained that are dealt with independently. All methods produce an output sequence

$$O = \langle (d, [t'_1, t'_2], p'_1), \dots, (d, [t'_m, t'_{m+1}], p'_m) \rangle \quad (3.18)$$

that consists of $|O| = m \leq n$ coalesced postings and covers the same time interval as the original postings from the input sequence I , so that $t_1 = t'_1$ and $t_{n+1} = t'_{m+1}$.

Our temporal coalescing techniques deal with postings belonging to a term v in isolation and therefore naturally allow for parallelization or implementation using a modern data-processing approach such as MapReduce [DG10]. We further assume that input sequences have already been computed. Letting L_v denote the list of all postings belonging to term v , computing the input sequences, i.e., sorting L_v and splitting it up into sequences of temporally adjacent postings is possible in time $O(|L_v| \cdot \log |L_v|)$.

3.6.1 Boolean Payloads

If only Boolean queries need to be supported, payloads are empty, as we laid out in Chapter 2. Postings in TTIIX thus have the form

$$(d, [t_i, t_j]),$$

indicating that a term was present in document d during the time interval $[t_i, t_j]$. Given an input sequence I , our objective is to determine a *minimal length* output sequence O that correctly captures when the term was present in the document. This can be formally stated as the following optimization problem:

Definition 3.6 (Boolean payload temporal-coalescing problem)

$$\underset{\mathcal{O}}{\operatorname{argmin}} |\mathcal{O}| \quad s.t.$$

$$\bigcup_{(d, [t_i, t_j]) \in I} [t_i, t_j) = \bigcup_{(d, [t'_i, t'_j]) \in \mathcal{O}} [t'_i, t'_j)$$

Since the input sequence I represents a contiguous time interval during which the term was present, as explained above, we can easily determine an optimal solution to the above optimization problem as

$$\mathcal{O} = \langle (d, [t_1, t_{n+1})) \rangle. \quad (3.19)$$

Our output hence contains only a single posting capturing that the term was present in the document during the contiguous time interval covered by the input sequence. Determining the output sequence is possible in time $O(n)$ – we still have to read the entire input sequence. The space complexity is in $O(1)$ – for keeping the coalesced posting.

As a side remark, note that the temporal coalescing for Boolean payloads just described can be seen as applying temporal coalescing as proposed for temporal databases and discussed in Chapter 2.

3.6.2 Scalar Payloads

Many changes to documents are minor, leaving large portions of the document untouched. For scalar posting payloads that capture, for instance, Okapi BM25 tf-scores or plain term frequencies such minor changes result in postings for consecutive document versions whose scalar payloads differ only slightly or not at all. Our temporal coalescing technique for scalar payloads reduces the number of postings in the index by coalescing sequences of postings belonging to consecutive document versions that bear *almost identical* scalar payloads. This idea is illustrated in Figure 3.3, which shows non-coalesced postings and coalesced postings belonging to a single document. In the example, temporal coalescing exploits the fact that the scalar payloads of postings differ only slightly and reduces the number of postings from nine to three. As can be seen from the figure, the technique introduces an approximation, since the scalar payloads of coalesced postings deviate from scalar payloads of non-coalesced postings.

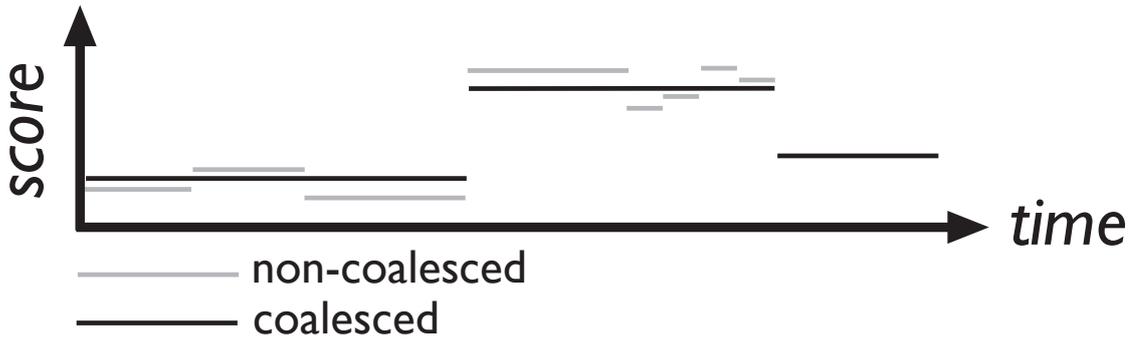


Figure 3.3: Temporal coalescing for scalar payloads illustrated

However, as we demonstrate in our experimental evaluation, this approximation hardly affects query results in practice.

We now formalize the problem addressed by temporal coalescing for scalar payloads. We are given an input sequence I as described above. Our objective is to generate a *minimal length* output sequence

$$O = \langle (d, [t'_1, t'_2], p'_1), \dots, (d, [t'_m, t'_{m+1}], p'_m) \rangle \quad (3.20)$$

that adheres to the following constraint. The relative approximation error induced by temporal coalescing must be less than or equal to a user-defined threshold ϵ . In detail, for two postings $(d, p_i, [t_i, t_{i+1}]) \in I$ and $(d, p'_j, [t'_j, t'_{j+1}]) \in O$, we demand

$$[t_i, t_{i+1}] \subseteq [t'_j, t'_{j+1}] \Rightarrow |p_i - p'_j| / |p_i| \leq \epsilon. \quad (3.21)$$

We thus address the following formal optimization problem:

Definition 3.7 (Scalar payload temporal-coalescing problem)

$$\operatorname{argmin}_O |O| \quad s.t.$$

$$\forall [t_i, t_{i+1}] \in I \quad \forall [t'_j, t'_{j+1}] \in O : [t_i, t_{i+1}] \subseteq [t'_j, t'_{j+1}] \Rightarrow |p_i - p'_j| / |p_i| \leq \epsilon.$$

Finding such a minimal length output sequence O can be cast into finding a piecewise-constant representation for the points (t_i, p_i) that uses a minimal number of segments while retaining the above approximation guarantee. Similar problems occur in time-series segmentation [KCHP01, TT06] and histogram construction [IP95, JKM⁺98]. In these settings, however, the typical objective is to minimize a global measure of error (e.g., the sum of squared errors) while using no more than B segments. An optimal solution to this problem can be obtained in $O(n^2 \cdot B)$ [JKM⁺98, TT06] time.

In our setting, as a key difference, only a guarantee on the local error needs to be retained – in contrast to a guarantee on the global error in the aforementioned settings. Exploiting this fact, we can compute an optimal solution to our problem using the greedy Algorithm 3.

Algorithm 3 performs one pass over the input sequence I . While doing so, the algorithm identifies maximal-length sequences of postings that can be coalesced into one posting without violating the above approximation guarantee.

In detail, for the coalesced posting currently being built, the algorithm maintains its valid-time interval $[t_b, t_e)$ and the range $[P_{low}, P_{high}]$ of legitimate choices for its scalar payload. When reading a new posting from the input I , the algorithm has to test whether the coalesced posting currently being built can be extended to cover the posting just read. To this end, the algorithm determines the range

$$[p_{low}, p_{high}] = [p_j - \epsilon \cdot |p_j|, p_j + \epsilon \cdot |p_j|]$$

that includes all values the scalar payload of a coalesced posting can assume without violating our approximation guarantee for the posting read. If

$$[P_{low}, P_{high}] \cap [p_{low}, p_{high}] \neq \emptyset$$

holds, the coalesced posting currently being built is extended by updating its valid-time interval and the range of legitimate scalar payloads. Otherwise, the coalesced posting is output using $\frac{1}{2} \cdot (P_{low} + P_{high})$ as its scalar payload, and a new coalesced posting is started based on the posting just read. Note that the choice $\frac{1}{2} \cdot (P_{low} + P_{high})$ is arbitrary – any value in $[P_{low}, P_{high}]$ is guaranteed not to violate the approximation guarantee for any of the postings that it represents.

The time complexity of the algorithm is in $O(n)$ – each of the n input postings is processed in time $O(1)$. Its space complexity is in $O(1)$ – for maintaining the information about the current coalesced posting. The following theorem states the optimality of Algorithm 3.

Theorem 3.2 *Algorithm 3 produces an optimal output sequence O .*

For the proof of the above theorem let \hat{O} denote an optimal output sequence, and let O denote the output produced by Algorithm 3. Further, let \hat{t}'_{b_i} (\hat{t}'_{e_i}) and t'_{b_i} (t'_{e_i}) refer to the left (right) valid-time interval boundary of the i -th postings in \hat{O} and O , respectively. The subsequence of I containing the i -th throughout the l -th posting is referred to as $I_{i..l}$ and we call it *coalescable* if a single coalesced posting retaining our approximation guarantee can be produced.

Algorithm 3: Temporal coalescing for scalar payloads

Data: Input sequence I and user-defined threshold ϵ

Result: Output sequence O

```

1 /* Initialization */
2  $O = \emptyset$ 

3 /* Information about coalesced posting currently being built */
4  $[t_b, t_e) = [t_1, t_2)$ 
5  $[P_{low}, P_{high}] = [p_1 - \epsilon \cdot |p_1|, p_1 + \epsilon \cdot |p_1|]$ 

6 /* Process postings from input sequence  $I$  */
7 for  $(d, p_i, [t_i, t_{i+1})) \in I$  do
8    $[p_{low}, p_{high}] = [p_i - \epsilon \cdot |p_i|, p_i + \epsilon \cdot |p_i|]$ 
9   if  $[P_{low}, P_{high}] \cap [p_{low}, p_{high}] \neq \emptyset$  then
10      $[P_{low}, P_{high}] = [P_{low}, P_{high}] \cap [p_{low}, p_{high}]$ 
11      $[t_b, t_e) = [t_b, t_{i+1})$ 
12   else
13     /* Output coalesced posting */
14      $O = O \cup \{(d, \frac{1}{2} \cdot (P_{low} + P_{high}), [t_b, t_e))\}$ 
15     /* Start new coalesced posting */
16      $[t_b, t_e) = [t_i, t_{i+1})$ 
17      $[P_{low}, P_{high}] = [p_i - \epsilon \cdot |p_i|, p_i + \epsilon \cdot |p_i|]$ 

18 /* Output last coalesced posting */
19  $O = O \cup \{(d, \frac{1}{2} \cdot (P_{low} + P_{high}), [t_b, t_e))\}$ 

```

We need two lemmas for the proof of the above theorem. The first lemma states that every subsequence of a coalescable sequence is itself coalescable.

Lemma 3.3 *If $I_{i..l}$ is coalescable all $I_{j..k}$ with $i \leq j \leq k \leq l$ are coalescable.*

Proof of Lemma 3.3 *By construction. Let $(d, [t_i, t_{l+1}], p')$ be the posting in O that represents $I_{i..l}$. Using the same value p' we can construct a posting $(d, [t_j, t_{k+1}], p')$ that retains the approximation guarantee on $I_{j..k}$. \square*

The second lemma states that the solution produced by Algorithm 3 stays ahead of an optimal solution.

Lemma 3.4 *Given \hat{O} and O as defined above, $\hat{t}'_{e_i} \leq t'_{e_i}$ holds*

Proof of Lemma 3.4 *By induction on i .*

$i = 1$: $\hat{t}'_{e_1} = t_j$ is the right time-interval boundary of the first posting in \hat{O} that covers the subsequence $I_{1..j}$, which is therefore known to be coalescable. Algorithm 3, by its greedy nature, will pick $I_{1..j}$ or a larger subsequence $I_{1..k}$ with $j \leq k$, so that $\hat{t}'_{e_1} \leq t'_{e_1}$.

$i \rightarrow i+1$: The $(i+1)$ -st posting produced by the optimal competitor has time-interval boundaries $\hat{t}'_{b_{i+1}}$ and $\hat{t}'_{e_{i+1}}$ and represents $I_{j..k}$. From $\hat{t}'_{e_i} \leq t'_{e_i}$ we know $\hat{t}'_{b_{i+1}} \leq t'_{b_{i+1}}$. Therefore, by Lemma 3.3 and its greedy nature, Algorithm 3 outputs a posting covering at least $I_{l..k}$ with $j \leq l$, which implies that $\hat{t}'_{e_{i+1}} \leq t'_{e_{i+1}}$. \square

Proof of Theorem 3.2 *By contradiction. Let us assume that $|\hat{O}| < |O|$. Since \hat{O} covers the whole input sequence, we know that $\hat{t}'_{e_{|\hat{O}|}} = t_{n+1}$. By Lemma 3.4 we know that $\hat{t}'_{e_{|\hat{O}|}} \leq t'_{e_{|\hat{O}|}}$ implying that $t'_{e_{|\hat{O}|}} = t_{n+1}$, since t_{n+1} is the right time-interval boundary of the last posting in the input. Thus, O has covered the whole input using $|\hat{O}|$ postings – a contradiction to our assumption. \square*

3.6.3 Positional Payloads

We now turn our attention to positional payloads that are required, for example, to support phrase queries or proximity scoring as explained in Chapter 2.

When not applying any temporal coalescing, postings have the form

$$(d, [t_i, t_j], \langle o_1, \dots, o_m \rangle),$$

where d and $[t_i, t_j)$ are a document identifier and a valid time-interval as before, and $\langle o_1, \dots, o_m \rangle$ is a sequence of word positions (also known as word offsets). Depending on implementation choices, each of the three components is represented compactly using one of the space-efficient encoding schemes discussed in Chapter 2.

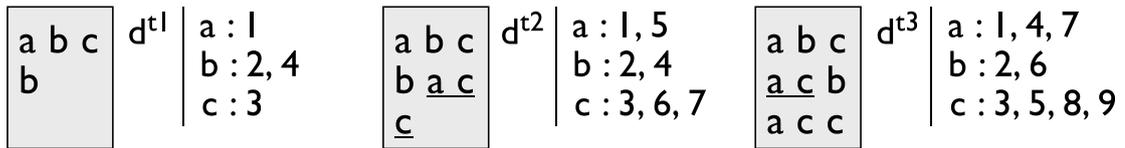


Figure 3.4: Document versions d^{t_1} , d^{t_2} , and d^{t_3} containing words a , b , and c .

In our concrete implementation, valid-time interval boundaries and document identifiers are encoded using 7-Bit encoding, word positions are compressed by encoding their gaps using 7-Bit encoding. For the problem instance shown in Figure 3.4, we obtain the following posting for document version d^{t_1} and word b

$$(d, [t_1, t_2), \langle 2, 4 \rangle),$$

Assuming that $d = 1317$, $t_1 = 1$, and $t_2 = 2$, storing this posting requires 7 bytes in our implementation.

On closer examination, Figure 3.4 reveals some key characteristics of changes in versioned document collections. First, as mentioned before, there is a high degree of redundancy among consecutive versions of the same document. Looking at the word b in Figure 3.4, for instance, we see that it occurs at the same positions in document versions d^{t_1} and d^{t_2} . Comparing the two document versions, we see that content was only appended, leaving all existing word positions untouched. Changes that add content to the end of a document are common in practice. Consider, as an example, bulletin boards on the Web where replies to a post are typically presented in chronological order. A second more subtle characteristic can be observed when comparing the positions of word c between document versions d^{t_2} and d^{t_3} . Although the two document versions only have a single word position in common, we see that word positions are numerically close. Positions $\{8, 9\}$ in the newer document version are just shifted equivalents of $\{6, 7\}$ in the old document version. When merging the two sequences, as a result of the numerical closeness, the resulting sequence $\{3, 5, 6, 7, 8, 9\}$ has small gaps between its elements. These smaller gaps in turn result in a better

compressibility of the merged sequence. Small shifts by a fixed margin are another common kind of change occurring in practice, for instance, when adding missing words or removing superfluous ones in a document.

The two observations presented above suggest that we can create a more compact representation of word positions by constructing postings that cover multiple consecutive document versions. We therefore extend the above posting structure to hold word positions for multiple consecutive document versions as follows

$$(d, [t_i, t_j], \langle o_1, \dots, o_m \rangle, \langle \sigma_{1,1}, \dots, \sigma_{1,m} \mid \dots \mid \sigma_{n,1}, \dots, \sigma_{n,m} \rangle).$$

The valid-time interval $[t_i, t_j)$ now covers the valid-time intervals of all n covered document versions. Similarly, the third component $\langle o_1, \dots, o_m \rangle$ keeps the sequence of all m positions where the word occurs in any of the document versions. In the fourth component, for each of the n covered document versions, a bit signature $\sigma_{i,1}, \dots, \sigma_{i,m}$ of length m is kept with the j -th bit conveying whether the j -th position is present in the i -th document version.

Again, as for the above original posting structure, there are different alternatives to represent the components. Our implementation represents valid-time interval boundaries and the document identifier explicitly using 7-Bit encoding. The sequences of positions is compressed by encoding their gaps using 7-Bit encoding. The signatures are stored explicitly, thus requiring $n \cdot m$ bits in our implementation.

For the problem instance in Figure 3.4 we obtain the following posting for b when covering all three document versions

$$(d, [t_1, \text{now}), \langle 2, 4, 6 \rangle, \langle 110 \mid 110 \mid 101 \rangle).$$

Making the same assumptions as above and assuming $\text{now} = 4$, storing this posting requires 18 bytes in our implementation. In contrast, storing three separate postings requires 21 bytes. Thus, by building the coalesced posting we save up to 14.2% of space.

As we argued above, one can often save significant space by building postings that cover multiple consecutive document versions. Depending on how the posting lists are internally organized, these coalesced postings may affect query processing adversely. To illustrate this, let us assume that we need to retrieve the word positions for a single document version, for instance, while processing a time-point phrase query. When we consider a naïve index organization and

query processing over it, the full list is read to retrieve the single posting. In this case, we fully profit from the space savings achieved, since less data needs to be read from disk. However, typical high-performance search systems employ sophisticated index organizations that could include skip pointers [MZ96] to jump directly (or close) to the desired posting during query processing. In our running example, this sort of index organization means that, in the worst case we read the 18 bytes of the coalesced posting instead of only 7 bytes for the original posting – an overhead factor of 2.58.

Having illustrated the benefits and pitfalls of building postings that cover multiple consecutive document versions, we now describe a method to systematically determine which positional postings should be coalesced. Our method is tunable by a parameter η specifying the maximum acceptable *worst-case overhead factor* for query processing introduced above. It minimizes the total cost required to represent the word positions in the given document versions under the constraint that we never pay an overhead factor larger than the parameter η .

For a fixed term v and fixed document d , the method is applied to a sequence I of n versions d^{t_i} of the document containing the word. For ease of explanation we assume that the right valid-time interval boundary of the last document version d^{t_n} is given as t_{n+1} . Our input thus consists of a sequence containing n postings with positional payloads

$$I = \langle (d, [t_1, t_2), \langle o_{1,1}, \dots \rangle), \dots, (d, [t_n, t_{n+1}), \langle o_{n,1}, \dots \rangle) \rangle. \quad (3.22)$$

For the description of the method, we abstract from the concrete posting structure, and let $c(k, l)$ denote the cost required to represent the posting covering the document versions d^{t_i} with $t_k \leq t_i < t_l$. We make the natural assumption that the cost is monotonous, i.e.,

$$[t_i, t_j) \subseteq [t_k, t_l) \Rightarrow c(i, j) \leq c(k, l). \quad (3.23)$$

The method outputs a set \mathcal{C} of disjoint time intervals that cover $[t_1, t_{n+1})$. For each $[t_k, t_l) \in \mathcal{C}$ we build a posting covering document versions d^{t_i} with $t_k \leq t_i < t_l$. Formally, the method solves the following optimization problem:

Definition 3.8 (Positional payload temporal-coalescing problem)

$$\begin{aligned} & \underset{\mathcal{C}}{\operatorname{argmin}} \sum_{[t_k, t_l) \in \mathcal{C}} c(k, l) \quad s.t. \\ & \forall [t_k, t_l) \in \mathcal{C} : [t_i, t_{i+1}) \subseteq [t_k, t_l) \Rightarrow c(k, l) \leq \eta \cdot c(i, i+1) \end{aligned}$$

Hence, the total cost to represent the word positions in the given document versions is minimized. The family of constraints ensures that we never pay an overhead factor larger than η when processing a query.

An optimal solution to the problem can be computed using dynamic programming based on the recurrence

$$\text{OPT}(k) = \min \left(v(1, k), \min_{1 < j < k} \text{OPT}(j) + v(j, k) \right),$$

where $v(j, k)$ reflects whether a single posting can cover document versions with a timestamp in $[t_j, t_k)$ without violating the above constraints on the overhead factor and is defined as

$$v(j, k) = \begin{cases} c(j, k) & : [t_i, t_{i+1}) \subseteq [t_j, t_k) \Rightarrow c(j, k) \leq \eta \cdot c(i, i+1) \\ \infty & : \text{otherwise} \end{cases}.$$

The recurrence captures that an optimal solution to the prefix subproblem $[t_1, t_k)$ either consists of (i) a single coalesced posting that covers $[t_1, t_k)$ or (ii) an optimal solution to a smaller prefix subproblem $[t_1, t_j)$ and a single coalesced posting that covers $[t_j, t_k)$. Algorithm 4 shows pseudo-code for computing an optimal solution. In the pseudo-code, we make use of the function v defined above and a function `coalesce` that produces a coalesced posting.

The algorithm computes optimal solutions for prefix subproblems $[t_1, t_k)$ of increasing length. For each of them, the algorithm examines combinations of an optimal solution to the prefix subproblem $[t_1, t_j)$ and a single coalesced posting covering the time interval $[t_j, t_k)$, as well as the solution that consists of a single coalesced posting covering the time interval $[t_1, t_k)$. In each iteration, the validity of the coalesced posting considered is checked – once this validity test fails, the algorithm terminates its inner loop early, which is possible because of the assumed cost monotonicity. The algorithm keeps track of the optimal cost for the prefix subproblem $[t_1, t_k)$ in `opt[k]`. In `split[k]` the left valid-time interval boundary of the rightmost partition in the optimal solution to the prefix subproblem $[t_1, t_k)$ is remembered. If `split[k] = j` the rightmost partition in the optimal solution to the prefix subproblem $[t_1, t_k)$ is thus $[t_j, t_k)$.

The algorithm has time complexity in $O(n^3)$ – in each iteration of the nested loop, the validity of a single coalesced posting covering the time interval $[t_j, t_k)$ is checked, which is possible in time $O(n)$. The space complexity is in $O(n)$ for keeping the costs of original postings and prefixes, as well as the splits. The cubic time complexity seems impractical first, nevertheless we found that the

Algorithm 4: Temporal coalescing for positional payloads (optimal)**Data:** Input sequence I and user-defined threshold η **Result:** Output sequence O

```

1 /* Initialization */
2 opt [2..n+1] =  $\langle \infty, \dots, \infty \rangle$ 
3 split [2..n+1] =  $\langle 0, \dots, 0 \rangle$ 
4 /* Dynamic programming */
5 for k = 2 to n + 1 do
6   for j = k - 1 to 1 do
7     cost = v(j, k)
8     if cost <  $\infty$  then
9       /* Update cost if this is a combined solution */
10      if j > 1 then
11        cost = opt[j] + cost
12      if cost < opt[k] then
13        opt[k] = cost
14        split[k] = j
15      else
16        break
17 /* Assemble optimal output sequence */
18 O =  $\emptyset$ 
19 e = n + 1
20 repeat
21   b = split[e]
22   O = { coalesce(b, e) }  $\cup$  O
23   e = b
24 until e = 1 ;

```

inner loop often terminates early for values of η that are close to 1.0, rendering the algorithm well-applicable in practice.

For large document collections or large values of η , one can resort to the greedy Algorithm 5. The algorithm considers postings in the input sequence I from left to right, i.e., in temporal order. While doing so, it builds coalesced postings greedily. When reading the next posting from the input sequence I , the algorithm checks whether the coalesced posting currently being built, which is represented by b and e , can be extended to include the posting just read. If this is not the case, a coalesced posting is appended to the output sequence O . Following that, a new coalesced posting is started by re-initializing the values of b , e , and minCost .

Algorithm 5 has time complexity in $O(n^2)$ – in each of the n iterations the cost $c(b, e)$ is computed in time $O(n)$. Its space complexity is in $O(1)$ – for keeping b , e , and minCost .

Despite its simplicity, the greedy Algorithm 5 comes with an approximation guarantee as we explain next. Let \mathcal{C} be the set of time intervals that corresponds to the solution determined by the greedy algorithm. That is, the time interval $[t_k, t_l) \in \mathcal{C}$ indicates that a coalesced posting is produced for all postings $(d, [t_i, t_{i+1}), \langle o_{i,1}, \dots \rangle)$ with $[t_i, t_{i+1}) \subseteq [t_k, t_l)$ from the input sequence. Analogously, we let \mathcal{C}^* denote the set of time intervals corresponding to an optimal solution. Recall from our description of the optimal algorithm above that $c(k, l)$ is the cost of a coalesced posting covering $[t_k, t_l)$. The following theorem states the greedy algorithm's approximation guarantee.

Theorem 3.3 *Algorithm 5 is a η -approximation algorithm. For the sets of time intervals \mathcal{C} and \mathcal{C}^* corresponding to the solution determined by Algorithm 5 and an optimal solution the following holds:*

$$\sum_{[t_k, t_l) \in \mathcal{C}} c(k, l) \leq \eta \cdot \sum_{[t_i, t_j) \in \mathcal{C}^*} c(i, j).$$

For the proof of Theorem 3.3 we need the following lemma:

Lemma 3.5 *For each time interval $[t_i, t_j) \in \mathcal{C}^*$ there is at most one time interval $[t_k, t_l) \in \mathcal{C}$ such that $t_k \in [t_i, t_j)$.*

Proof of Lemma 3.5 *By contraction. Assume that there is a $[t_i, t_j) \in \mathcal{C}^*$ and two consecutive time intervals $[t_k, t_l)$ and $[t_l, t_m)$ from \mathcal{C} such that $t_k \in [t_i, t_j)$ and*

Algorithm 5: Temporal coalescing for positional payloads (approximate)

Data: Input sequence I and user-defined threshold η **Result:** Output sequence O

```
1 /* Initialization */
2  $O = \emptyset$ 
3  $b = 1$ 
4  $\text{minCost} = c(1, 2)$ 
5 /* Determine postings to be coalesced greedily */
6 for  $e = 2$  to  $n + 1$  do
7    $\text{minCost} = \min(\text{minCost}, c(e - 1, e))$ 
8    $\text{cost} = c(b, e)$ 
9   if  $\text{cost} > \eta \cdot \text{minCost}$  then
10      $O = O \cup \{\text{coalesce}(b, e - 1)\}$ 
11      $b = e - 1$ 
12      $\text{minCost} = c(e - 1, e)$ 
13  $O = O \cup \{\text{coalesce}(b, e)\}$ 
```

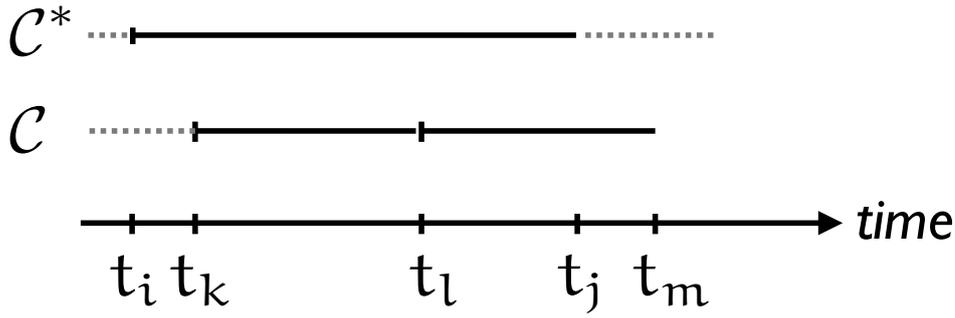


Figure 3.5: Situation impossible according Lemma 3.5

$t_l \in [t_i, t_j)$ – the situation that Figure 3.5 depicts. By virtue of Algorithm 5’s greedy nature we know that the coalesced posting corresponding to $[t_k, t_l) \in \mathcal{C}$ can not be extended without violating our constraint on the overhead factor. The assumed optimal solution, however, contains a coalesced posting for $[t_i, t_j)$ with $t_i \leq t_k < t_l < t_j$ and must therefore, as a result of the assumed cost monotonicity, violate our constraint on the overhead factor. Consequently, \mathcal{C}^* is not an optimal solution – a contradiction to our initial assumption. \square

Proof of Theorem 3.3 By construction. Since both \mathcal{C} and \mathcal{C}^* cover the entire time interval $[t_1, t_{n+1})$, for each $[t_k, t_l) \in \mathcal{C}$ there is a $[t_i, t_j) \in \mathcal{C}^*$ such that $t_k \in [t_i, t_j)$. Moreover, from Lemma 3.5, we know that $[t_k, t_l)$ is the only such time interval from \mathcal{C} . Further, as a consequence of our constraint on the overhead factor, we know that $c(k, l) \leq \eta \cdot c(i, j)$ and can thus write

$$\sum_{[t_k, t_l) \in \mathcal{C}} c(k, l) \leq \eta \cdot \sum_{[t_i, t_j) \in \mathcal{C}^*} c(i, j).$$

\square

When processing time-point queries, the coalesced postings produced by the techniques described above can be used without modifying the query processing. As stated earlier, the result of a time-point query contains at most one version per document, so that the document identifier is sufficient to identify a result document version. The result of a time-interval query, in contrast, may contain more than one version per document. In that case, a combination of document identifier and timestamp is necessary to identify a result document version. Our coalesced postings, however, only keep track of an overall valid-time interval, but do not record the timestamps of coalesced versions. When

processing time-interval queries, we therefore keep a *document-version dictionary* in main memory that for each document contains the timestamps of its versions in sorted order. Using the document-version dictionary, we can efficiently identify the document versions represented by a coalesced posting that we read from a posting list.

The temporal coalescing techniques presented in this section are highly effective means to reduce the number of postings in a TTX instance and thus its overall space consumption, as we demonstrate empirically in Section 3.10. The following section deals with partitioning strategies that allow fine-tuning TTX with regard to performance requirements or space constraints.

3.7 Partitioning Strategies

In this section, we address the issue of determining at index-build time which posting lists $L_v : [t_k, t_l)$ should be kept in our index for term v , i.e., partitioning the list L_v by computing the set \mathcal{P}_v . When keeping multiple posting lists per term, we systematically replicate postings whose valid-time interval overlaps with multiple time intervals associated with posting lists, thus increasing the size of our index.

At a first glance, it may seem counterintuitive to reduce index size using temporal coalescing, and then to increase it again by replicating postings using the partitioning strategies presented in this section. However, our main objective is to improve query-processing performance, not to reduce the index size alone. The use of temporal coalescing improves the performance by reducing the index size, while the temporal partitioning improves performance by judiciously replicating postings. Further, the two techniques can be applied separately and are independent. If applied in conjunction, though, there is a synergetic effect – posting lists in a temporally coalesced index are generally smaller.

Our methods proposed in this section focus primarily on time-point queries. However, time-interval queries can still be processed based on the determined partitionings. For some of our partitioning strategies this is possible even with theoretical guarantees, as we describe when applicable. In practice, we can always efficiently process time-interval queries, as our experimental evaluation in Section 3.10 demonstrates.

To aid the presentation in the rest of this section, we first provide some definitions. Let

$$T_v = \langle t_1 \dots t_n \rangle \quad (3.24)$$

be the sorted sequence of all unique time-interval boundaries in L_v . Note that $|T_v| \leq 2 \cdot |L_v|$, so that $O(n) \subseteq O(|L_v|)$. Moreover, we define

$$\mathcal{E}_v = \{ [t_i, t_{i+1}) \mid 1 \leq i < n \} \quad (3.25)$$

to be the set of *elementary time intervals*. These are elementary in the sense that all $t \in [t_i, t_{i+1})$ share the same set of alive postings. As stated earlier, we refer to the *partitioning*, i.e., the set of time intervals for which posting lists are kept in our index as

$$\mathcal{P}_v \subseteq \{ [t_i, t_j) \mid 1 \leq i < j \leq n \}, \quad (3.26)$$

and demand that it completely covers the time interval $[t_1, t_n)$, i.e.,

$$\bigcup_{[t_k, t_l) \in \mathcal{P}_v} [t_k, t_l) = [t_1, t_n). \quad (3.27)$$

The *space* for keeping posting lists with associated time intervals in \mathcal{P}_v is

$$S(\mathcal{P}_v) = \sum_{[t_k, t_l) \in \mathcal{P}_v} |L_v : [t_k, t_l)|, \quad (3.28)$$

i.e., the overall number of postings contained in the posting lists that we keep for the term v in our index according to \mathcal{P}_v .

The *processing cost* for a time-point keyword query with time point t is

$$PC(t) = |L_v : [t_k, t_l)|, \quad (3.29)$$

where $L_v : [t_k, t_l)$ is the shortest posting list such that $t \in [t_k, t_l)$ as explained in Section 3.5. Analogously, for a query time-interval $[t_b, t_e]$ we define its processing cost as

$$PC([t_b, t_e]) = |L_v^\triangleright : [t_{b_1}, t_{e_1})| + \sum_{[t_{b_i}, t_{e_i}) \in \mathcal{L}_v} |L_v^+ : [t_{b_i}, t_{e_i})| \quad (3.30)$$

where \mathcal{L}_v is determined as described in Section 3.5. Our definition of processing cost thus accounts for the total number of postings read and inversely reflects the *query-processing performance* that we achieve.

3.7.1 Performance-Optimal Approach

As discussed in Section 3.5, the performance when processing a time-point query q^t on a TTX instance is influenced adversely by the wasted I/O due to read but filtered-out postings. Temporal coalescing implicitly addresses this problem by reducing the number of postings and the space consumption of the posting lists scanned, but still a significant overhead remains. We now tackle this problem and describe temporal partitioning strategies geared at time-point queries that determine for each term v a set of posting lists that should be kept in the index.

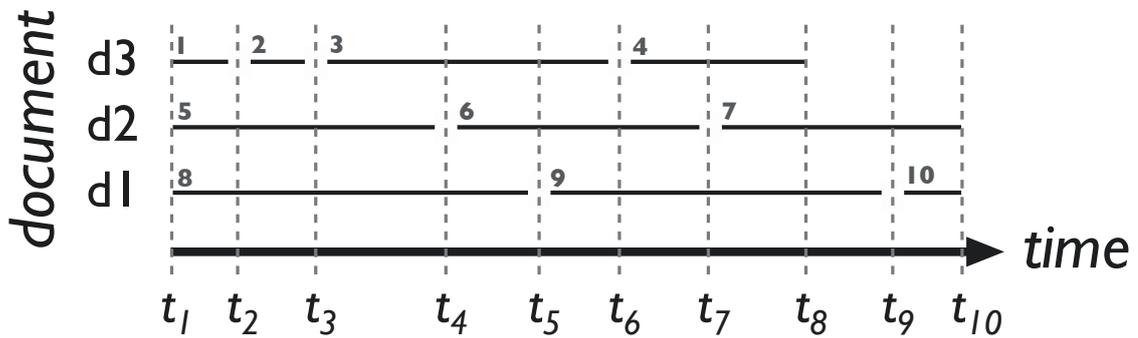


Figure 3.6: Partitioning illustrated

We illustrate the trade-offs of temporal partitioning for time-point queries using the example given in Figure 3.6. The figure shows a total of ten postings belonging to term v and three different documents d_1 , d_2 , and d_3 . For ease of description, we have numbered boundaries of valid-time intervals in increasing time-order, as t_1, \dots, t_{10} and numbered the postings themselves as $1, \dots, 10$. Now, consider that we want to process a time-point query with $t \in [t_1, t_2)$. Only three postings (namely, 1, 5, and 8) are valid at time t and therefore required to process the query. In the worst case, if only a single list $L_v : [t_1, t_{10})$ is contained in our index, we have to read all ten postings. In the best case, if the list $L_v : [t_1, t_2)$ is contained in the index, we achieve the optimal query-processing performance, reading only the three postings required to answer the query.

This last observation suggests one strategy to eliminate the problem of filtered-out postings entirely. By choosing

$$\mathcal{P}_v = \mathcal{E}_v \quad (3.31)$$

and thus keeping a posting list for every elementary time interval, for any query time-point t only the postings valid at that time are read, so that the optimal

query-processing performance is achieved. Also, for a time-interval query $q^{[t_b, t_e]}$ every posting whose valid-time interval overlaps with the query time-interval $[t_b, t_e]$ is read exactly once – the approach thus reads the minimal number of postings and achieves optimal query-processing performance.

Subsequently, we refer to this performance-optimal approach as P_{opt} . However, note that P_{opt} induces a significant blow-up in space consumption. When applied to the scenario shown in Figure 3.6, P_{opt} leads to an index that contains 25 postings in total.

3.7.2 Space-Optimal Approach

If space is at a premium and we want to keep the space consumption of the index as small as possible, we can choose

$$\mathcal{P}_v = \{ [t_1, t_n) \}, \quad (3.32)$$

keeping only a single posting list for the term in the index. This approach achieves the minimal space consumption, since every posting is kept exactly once in the index – no replication is performed. It does not achieve good query-processing performance, as we argued above. This space-optimal approach will subsequently be referred to as S_{opt} .

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

3.7.3 Performance-Guarantee Approach

The P_{opt} approach clearly wastes a lot of space keeping many nearly-identical posting lists. In the example illustrated in Figure 3.6 posting lists for $[t_1, t_2)$ and $[t_2, t_3)$ differ only by one posting. If a posting list for $[t_1, t_3)$ was kept instead, one could save significant space while incurring only an overhead of one filtered-out posting for all $t \in [t_1, t_3)$. The technique presented next is driven by the idea that significant space savings over P_{opt} are achievable, if an upper-bounded loss on the performance can be tolerated, or to put it differently, if a

performance guarantee relative to the optimum must be retained. In detail, the technique, which we refer to as PG (Performance Guarantee) in the remainder, determines a set \mathcal{P}_v that consumes minimal space, but guarantees for any query time-point $t \in [t_1, t_n)$ that the query-processing cost is worse than optimal by at most a factor of $\gamma \geq 1$, i.e., at most a fraction of $(\gamma - 1)$ superfluous postings are read. The underlying optimization problem can be formally defined as follows:

Definition 3.9 (Performance-guarantee partitioning problem)

$$\operatorname{argmin}_{\mathcal{P}_v} S(\mathcal{P}_v) \quad s.t.$$

$$\forall t \in [t_1, t_n) : PC(t) \leq \gamma \cdot |L_v : t| ,$$

where $L_v : t$ is the posting list containing all postings from L_v that are alive at time t , so that $|L_v : t|$ is the number of postings required to process a time-point keyword query with time t .

We can compute an optimal solution to this optimization problem based on the following recurrence

$$\text{OPT}(k) = \min \left(v(1, k), \min_{1 < j < k} (\text{OPT}(j) + v(j, k)) \right) ,$$

where $v(j, k)$ is defined as

$$v(j, k) = \begin{cases} |L_v : [t_j, t_k)| & : \quad \forall t \in [t_j, t_k) : |L_v : [t_j, t_k)| \leq \gamma \cdot |L_v : t| \\ \infty & : \quad \text{otherwise} \end{cases}$$

and captures whether $L_v : [t_j, t_k)$ retains the above performance guarantee whenever it is employed to process a time-point keyword query.

Intuitively, the recurrence states that an optimal solution for the prefix subproblem $[t_1, t_k)$ either consists of a single valid posting list $L_v : [t_1, t_k)$ or can be combined from an optimal solution to a smaller prefix subproblem $[t_1, t_j)$ and a valid posting list $L_v : [t_j, t_k)$. Only valid posting lists that do not violate the performance guarantee can thus be part of an optimal solution.

Algorithm 6 evaluates the above recurrence by looking at prefix subproblems $[t_1, t_k)$ of increasing length. For the prefix subproblem $[t_1, t_k)$, the algorithm checks whether keeping the single posting list $L_v : [t_1, t_k)$ yields a valid solution that retains the performance guarantee. If this is not the case, the algorithm determines the optimal solution combining an optimal solution to a smaller prefix subproblem $[t_1, t_j)$ and a single posting list $L_v : [t_j, t_k)$. The algorithm keeps

Algorithm 6: Performance-guarantee partitioning (optimal)

Data: List L_v with $T = \{t_1, \dots, t_n\}$ and user-defined threshold γ

Result: Optimal partitioning \mathcal{P}_v

```

1 /* Initialization */
2 opt [2..n] =  $\langle \infty, \dots, \infty \rangle$ 
3 split [2..n] =  $\langle 0, \dots, 0 \rangle$ 
4 /* Dynamic programming */
5 for k = 2 to n do
6     for j = k - 1 to 2 do
7         cost = v(j, k)
8         if cost <  $\infty$  then
9             /* Update cost if this is a combined solution */
10            if j > 1 then
11                cost = opt[j] + cost
12            if cost < opt[k] then
13                opt[k] = cost
14                split[k] = j
15        else
16            break
17 /* Assemble partitioning  $\mathcal{P}_v$  */
18  $\mathcal{P}_v = \emptyset$ 
19 e = n
20 repeat
21     b = split[e]
22      $\mathcal{P}_v = \{[t_b, t_e]\} \cup \mathcal{P}_v$ 
23     e = b
24 until e = 1 ;

```

track of the optimal space $\text{OPT}(k)$ for the prefix subproblem $[t_1, t_k)$ in $\text{opt}[k]$ and records the left time-interval boundary of the rightmost partition in the optimal solution in $\text{split}[k]$. Thus, if $\text{split}[k] = j$ the rightmost partition in the optimal solution for the prefix subproblem $[t_1, t_k)$ is $[t_j, t_k)$.

The time and space complexities of Algorithm 6 depend on whether posting sizes $|L_v : [t_i, t_j)|$ have been precomputed, which can be done in time $O(|L_v| + n^2)$ and space $O(n^2)$. If this is the case, Algorithm 6 can be implemented to have time complexity in $O(|L_v| + n^2)$ – for each of the n prefix subproblem the above recurrence must be evaluated, which is then possible in time $O(n)$. Its space complexity is in $O(n^2)$ – the cost of keeping the precomputed posting-list lengths and memoizing optimal solutions to prefix subproblems. Otherwise, the algorithm has time complexity in $O(n^2 \cdot |L_v|)$, because of the additional $O(|L_v|)$ effort needed per iteration to compute the posting list size. Its space complexity, however, is then in $O(n)$ for keeping opt and split .

For large document collections, and therefore large $|L_v|$, these complexities can become prohibitive in practice. Fortunately, though, an approximate solution to the above optimization problem can be computed using an efficient greedy algorithm in time $O(|L_v|)$ and space $O(n)$, as we describe in the following.

Algorithm 7 provides pseudo-code of the greedy approximation algorithm. As a precomputation, the algorithm makes one pass over L_v and initializes two arrays `created` and `expires`. In the former `created[i]` records the number of postings created at time t_i ; in the latter `expires[i]` records the number of postings that expire at time t_i . Following that, the algorithm makes one pass over the time points in T and, while doing so, greedily constructs partitions. To this end, the algorithm keeps track of `length` as the number of postings in the partition currently being built, `currentAlive` as the number of postings alive at the current time point t_i , and `minAlive` as the minimal number of postings alive at any time point in the partition currently being build. When inspecting the next time point t_{i+1} , the algorithm checks whether the current partition can be extended to include t_{i+1} without violating the performance guarantee. If this is not the case, the partition is added to \mathcal{P}_v and a new partition is started. Interestingly, this greedy algorithm comes with an approximation guarantee:

Theorem 3.4 *Algorithm 7 is a 2-approximation algorithm, i.e., for the partitioning \mathcal{P}_v determined and an optimal partitioning \mathcal{P}_v^* the following holds:*

$$S(\mathcal{P}_v) \leq 2 \cdot S(\mathcal{P}_v^*).$$

Algorithm 7: Performance-guarantee partitioning (approximate)

Data: List L_v with $T = \{t_1, \dots, t_n\}$ and user-defined threshold γ

Result: Partitioning \mathcal{P}_v

```

1 /* Initialization */
2  $\mathcal{P}_v = \langle \rangle$ 
3  $\text{created}[1..n] = \langle 0, \dots, 0 \rangle$ 
4  $\text{expires}[1..n] = \langle 0, \dots, 0 \rangle$ 
5 for  $(d, p, [t_i, t_j]) \in L_v$  do
6   |  $\text{created}[i]++$ 
7   |  $\text{expires}[j]++$ 
8 /* Determine partitions greedily */
9  $t_b = t_1$ 
10  $\text{length} = 0$ 
11  $\text{currentAlive} = 0$ 
12  $\text{minAlive} = \infty$ 
13 for  $i = 2$  to  $n$  do
14   |  $\text{currentAlive}' = \text{currentAlive} - \text{expires}[i - 1] + \text{created}[i - 1]$ 
15   |  $\text{minAlive}' = \min(\text{minAlive}, \text{currentAlive}')$ 
16   |  $\text{length}' = \text{length} + \text{created}[i - 1]$ 
17   | if  $\text{length}' \leq \gamma \cdot \text{minAlive}'$  then
18     |  $\text{length} = \text{length}'$ 
19     |  $\text{currentAlive} = \text{currentAlive}'$ 
20     |  $\text{minAlive} = \text{minAlive}'$ 
21   | else
22     |  $\mathcal{P}_v = \mathcal{P}_v \cup \{[t_b, t_{i-1}]\}$ 
23     |  $\text{length} = \text{currentAlive}'$ 
24     |  $\text{minAlive} = \text{length}$ 
25     |  $t_b = t_{i-1}$ 
26 /* Add last partition */
27  $\mathcal{P}_v = \mathcal{P}_v \cup \{[t_b, t_n]\}$ 

```

The proof of the theorem is similar to our earlier proof of Theorem 3.3. As a stepping stone for the proof we need the following lemma:

Lemma 3.6 *For each partition $[t_i, t_l] \in \mathcal{P}_v^*$ there are at most two consecutive partitions $[t_j, t_k)$ and $[t_k, t_m)$ in \mathcal{P}_v such that $[t_i, t_l] \cap [t_j, t_k) \neq \emptyset$ and $[t_i, t_l] \cap [t_k, t_m) \neq \emptyset$.*

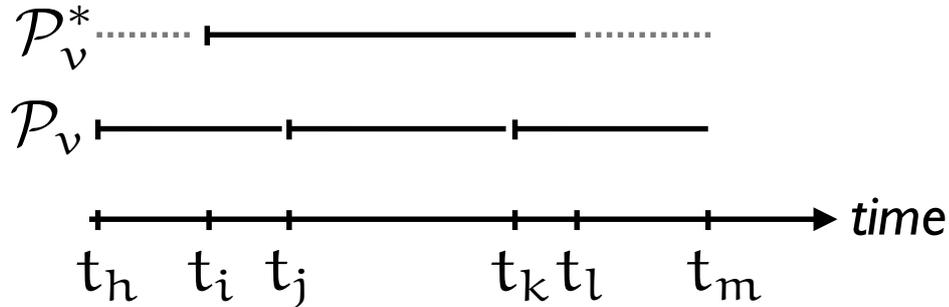


Figure 3.7: Situation impossible according to Lemma 3.6

Proof of Lemma 3.6 *By contradiction. Assume that there are three consecutive partitions $[t_h, t_j)$, $[t_j, t_k)$, and $[t_k, t_m)$ in \mathcal{P}_v so that each of them overlaps with $[t_i, t_l]$ – the situation that Figure 3.7 depicts. By Algorithm 7’s greedy nature we know that, when the second partition $[t_j, t_k)$ is output, there is no way to further extend it without violating our performance guarantee. Since*

$$|L_v : [t_i, t_l]| > |L_v : [t_j, t_k)| \quad (3.33)$$

there must be a $t \in [t_j, t_l]$ for which \mathcal{P}_v^ violates the performance guarantee – otherwise the greedy Algorithm 7 could have output $[t_j, t_l]$. As a consequence, \mathcal{P}_v^* can not be an optimal partitioning – a contradiction to our initial assumption. \square*

Proof of Theorem 3.4 *By construction. Both \mathcal{P}_v^* and \mathcal{P}_v cover the entire time interval $[t_1, t_n)$. Further, from Lemma 3.6, we know that for every posting contained in a partition from an optimal \mathcal{P}_v^* we keep at most two copies of it in the approximate \mathcal{P}_v . We can therefore write*

$$S(\mathcal{P}_v) = \sum_{[t_k, t_l] \in \mathcal{P}_v} |L_v : [t_k, t_l]| \leq 2 \cdot \sum_{[t_k, t_l] \in \mathcal{P}_v^*} |L_v : [t_k, t_l]| = 2 \cdot S(\mathcal{P}_v^*).$$

\square

Observe that Algorithm 6 and Algorithm 7 determine partitionings that consist of mutually disjoint time intervals covering $[t_1, t_n)$ – a fact that we leverage below when analyzing their behavior for processing time-interval queries.

As already hinted at in Chapter 2, our greedy approximation Algorithm 7 is equivalent to the algorithm proposed by Ramaswamy [Ram97]. Their partitioning technique therefore produces an approximate solution to the optimization problem underlying PG. Further, as shown in Ramaswamy [Ram97], for the determined partitioning

$$S(\mathcal{P}_v) \leq \frac{2 \cdot \gamma}{\gamma - 1} \cdot |L_v| \quad (3.34)$$

holds. Therefore, for our performance-guarantee approach, the total space required to keep the partitioned posting lists for term v in our index is in $O(|L_v|)$.

By definition, for time-point queries, our PG approach retains a performance guarantee, thus limiting the number of read but filtered-out postings. However, even when processing a time-interval query using a partitioning determined by Algorithm 6 or Algorithm 7, a performance guarantee is retained. Let $L_v : [t_b, t_e]$ be the posting list containing all postings from L_v with a valid-time interval $[t_i, t_j] \cap [t_b, t_e] \neq \emptyset$ – these are the postings that must be read to process a query with time-interval $[t_b, t_e]$.

Theorem 3.5 *For a partitioning \mathcal{P}_v that consists of disjoint time intervals covering $[t_1, t_n]$ and retains the performance guarantee for any $t \in [t_1, t_n]$, the following holds for a query with time-interval $[t_b, t_e]$: $PC([t_b, t_e]) \leq (2\gamma + 1) \cdot |L_v : [t_b, t_e]|$*

Proof of Theorem 3.5 *By construction. From our above observation we know that Algorithm 6 and Algorithm 7 determine partitionings consisting of disjoint time intervals that retain our performance guarantee for any query time-point. The sequence*

$$\mathcal{L}_v = \langle [t_{b_1}, t_{e_1}), \dots, [t_{b_m}, t_{e_m}) \rangle$$

determined for the query time-interval $[t_b, t_e]$ thus consists of disjoint time intervals. Further, it is easy to see that $t_b \in [t_{b_1}, t_{e_1})$ and $t_e \in [t_{b_m}, t_{e_m})$ must hold. The processing cost for the query time-interval $[t_b, t_e]$ is thus

$$\begin{aligned} & |L_v^\triangleright : [t_{b_1}, t_{e_1})| + |L_v^+ : [t_{b_1}, t_{e_1})| + \sum_{i=2}^{m-1} |L_v^+ : [t_{b_i}, t_{e_i})| + |L_v^+ : [t_{b_m}, t_{e_m})| \\ & \leq \gamma \cdot |L_v : t_b| + \sum_{i=2}^{m-1} |L_v^+ : [t_{b_i}, t_{e_i})| + \gamma \cdot |L_v : t_e| \\ & \leq \gamma \cdot |L_v : [t_b, t_e]| + |L_v : [t_b, t_e]| + \gamma \cdot |L_v : [t_b, t_e]| \\ & = (2\gamma + 1) \cdot |L_v : [t_b, t_e]| \end{aligned}$$

□

3.7.4 Space-Bound Approach

Thus far, we considered the problem of finding a partitioning that retains a guarantee on query-processing performance while consuming minimal space. In many situations, though, the storage space is at a premium and the aim would be to find a partitioning that optimizes an expected measure of query-processing performance while not exceeding a given space bound. The technique presented next, which is named SB (Space Bound), tackles this very problem. The space bound is modeled by means of a user-defined threshold $\kappa \geq 1$ that limits the maximum allowed blowup in index size from the space-optimal solution provided by S_{opt} . The SB technique seeks to find a partitioning \mathcal{P}_v that adheres to this space bound but minimizes the expected processing cost (and thus optimizes the expected performance). In the definition of the expected processing cost, $P(t)$ denotes the probability of the query time-point t . Formally, this space-bound partitioning problem is defined as follows:

Definition 3.10 (Space-bound partitioning problem)

$$\begin{aligned} \operatorname{argmin}_{\mathcal{P}_v} \quad & \sum_{t \in [t_1, t_n)} P(t) \cdot \text{PC}(t) \quad s.t. \\ & S(\mathcal{P}_v) \leq \kappa \cdot |L_v|. \end{aligned}$$

Before we explain how an optimal solution to this problem can be computed, let us introduce some definitions. We define the probability that a query time-point t falls into the time interval $[t_i, t_j)$ as

$$P([t_i, t_j)) = \sum_{t \in [t_i, t_j)} P(t). \quad (3.35)$$

For a partitioning \mathcal{P}_v we can hence rewrite its expected processing cost as

$$\text{EPC}(\mathcal{P}_v) = \sum_{[t_i, t_j) \in \mathcal{P}_v} P([t_i, t_j)) \cdot |L_v : [t_i, t_j)|. \quad (3.36)$$

The above optimization problem can then be solved using dynamic programming over prefix subproblems of increasing size and increasing space bounds using the recurrence

$$\text{OPT}(k, s) = \min \left(\begin{array}{c} P([t_1, t_k)) \cdot |L_v : [t_1, t_k)|, \\ \min_{1 < j < k} \text{OPT}(j, s - |L_v : [t_j, t_k)|) + P([t_j, t_k)) \cdot |L_v : [t_j, t_k)| \end{array} \right).$$

Algorithm 8: Space-bound partitioning (optimal)

Data: List L_v with $T = \{t_1, \dots, t_n\}$ and user-defined threshold κ

Result: Optimal partitioning \mathcal{P}_v

```

1 /* Initialization */
2 opt [2..n][0..⌊κ · |Lv⌋] = ⟨⟨∞, ..., ∞⟩, ..., ⟨∞, ..., ∞⟩⟩,
3 split [2..n][0..⌊κ · |Lv⌋] = ⟨⟨0, ..., 0⟩, ..., ⟨0, ..., 0⟩⟩
4 /* Dynamic programming */
5 for k = 2 to n do
6     for s = |Lv : [t1, tk]| to ⌊κ · |Lv⌋ do
7         /* Single posting list Lv : [t1, tk] */
8         opt [k][s] = P([t1, tk]) · |Lv : [t1, tk]|
9         split [k][s] = 1
10        /* Try to find a combined solution */
11        if k > 2 then
12            for j = k - 1 to 2 do
13                if opt[j][s - |Lv : [tj, tk]|] + P([tj, tk]) · |Lv : [tj, tk]| < opt[k][s]
14                    then
15                        opt[k][s] = opt[j][s - |Lv : [tj, tk]|] + P([tj, tk]) · |Lv : [tj, tk]|
16                        split[k][s] = j
17 /* Assemble optimal partitioning  $\mathcal{P}_v$  */
18  $\mathcal{P}_v = \langle \rangle$ 
19 s = ⌊κ · |Lv⌋
20 e = n
21 b = split[n][s]
22 while e > 1 do
23      $\mathcal{P}_v = \{[t_b, t_e]\} \cup \mathcal{P}_v$ 
24     s = s - |Lv : [tb, te]|
25     e = b
26     b = split[e][s]

```

According to this recurrence, the optimal (i.e., minimal) expected processing cost $\text{OPT}(k, s)$ for the prefix subproblem $[t_1, t_k)$ that retains a space bound s is obtained by either (i) keeping a single posting list $L_v : [t_1, t_k)$ or (ii) combining a posting list $L_v : [t_1, t_k)$ with the optimal solution to the prefix subproblem $[t_1, t_j)$ that consumes at most $s - |L_v : [t_j, t_k)|$ space.

Algorithm 8 gives pseudo-code for evaluating the recurrence using dynamic programming. In its outer loop, the algorithm examines prefix subproblems $[t_1, t_k)$ of increasing size. For each such prefix subproblem, the algorithm determines an optimal solution for all space bounds between $|L_v : [t_1, t_k)|$ and the global space bound $\lfloor \kappa \cdot |L_v| \rfloor$. Based on the above recurrence, the algorithm takes the partitioning consisting only of the posting list $L_v : [t_1, t_k)$ and then tries to find a better solution that combines an optimal solution to a prefix subproblem $[t_1, t_j)$ and a single posting list $L_v : [t_j, t_k)$. In the arrays `opt` and `split`, the algorithm memoizes the optimal expected processing cost and the left time-interval boundary of the rightmost partition. That is, `opt[k][s]` records the optimal expected processing cost for the prefix subproblem $[t_1, t_k)$ when using s space, and `split[k][s] = j` indicates that the rightmost partition in the corresponding optimal partitioning is $[t_j, t_k)$. Eventually, by backtracking through the information memoized in `split`, the algorithm assembles an optimal partitioning \mathcal{P}_v .

As for Algorithm 6, the time and space complexities of Algorithm 8 depend on whether posting list sizes $|L_v : [t_i, t_j)|$ have been precomputed, which is possible in time $O(|L_v| + n^2)$ and space $O(n^2)$. If this is the case, the time complexity of the algorithm is in $O(|L_v| \cdot n^2)$, and its space complexity is in $O(n \cdot |L_v|)$. Otherwise, if posting list lengths have not been precomputed, the time complexity is in $O(|L_v|^2 \cdot n^2)$, and the space complexity remains unchanged in $O(n \cdot |L_v|)$.

In practice, it is often the case that $|L_v| \gg n$, for instance, because of correlated document changes or coarse-grained timestamps at the granularity of days. Therefore, Algorithm 8 is not practical for large datasets, and we resort to an approximation in our implementation, which we describe in the following.

We employ simulated annealing [KJV83, KT05] to obtain an approximate solution to our problem. Pseudo-code for the computation is given in Algorithm 9. Simulated annealing takes a fixed number R of rounds to traverse the solution space of partitionings. The current partitioning is represented as a bit vector `boundaries` of length n with the i -th bit indicating whether two temporally adjacent partitions end and begin at t_i , respectively. Initially, all but the first and last bit are set to `false`, which corresponds to the partitioning $[t_1, t_n)$, i.e., the

Algorithm 9: Space-bound partitioning (approximate)

Data: List L_v with $T = \{t_1, \dots, t_n\}$, user-defined threshold κ , and number of rounds R

Result: Partitioning \mathcal{P}_v

```

1 /* Initialization */
2 boundaries =  $\langle 1, 0, \dots, 0, 1 \rangle$ 
3 bestBoundaries = boundaries
4 space =  $|L_v : [t_1, t_n]|$ 
5 EPC =  $|L_v : [t_1, t_n]|$ 
6 bestEPC = EPC

7 /* Simulated annealing */
8 for r = 1 to R do
9      $i = 1 + \lfloor (n - 2) \cdot \text{rnd}() \rfloor$ 
10    if space +  $|L_v : t_i| \leq \kappa \cdot |L_v|$  then
11        boundaries[i] =  $\neg$  boundaries[i]
12        EPC' = EPC(boundaries)
13        if EPC' < EPC  $\vee$  rnd() <  $e^{-\frac{|EPC - EPC'|}{R - r + 1}}$  then
14            EPC = EPC'
15            space = space +  $|L_v : t_i|$ 
16            if EPC < bestEPC then
17                bestEPC = EPC
18                bestBoundaries = boundaries
19        else
20            boundaries[i] =  $\neg$  boundaries[i]

21 /* Assemble partitioning  $\mathcal{P}_v$  */
22  $\mathcal{P}_v = \langle \rangle$ 
23 b = 1
24 for e = 2 to n do
25     if bestBoundaries[e] = 1 then
26          $\mathcal{P}_v = \mathcal{P}_v \cup \{[t_b, t_e]\}$ 
27         b = e

```

space-optimal partitioning that would be determined by S_{opt} . The first and the last bit in `boundaries` are always set to `true`. In each round, a random successor of the current partitioning is examined. This successor is generated by randomly flipping one of the $n - 2$ intermediate bits – the method `rnd()` return a random number in $[0, 1]$. If the successor does not adhere to the space bound, it is always rejected. If the successor adheres to the space bound, it is accepted if it improves the expected processing cost EPC – the method `EPC(boundaries)` returns the expected processing cost for the current assignment of `boundaries`. If it leads to higher expected processing cost as the current solution, it is accepted with probability $e^{-\frac{|EPC - EPC'|}{R - r + 1}}$ where $|EPC - EPC'|$ reflects the change in expected processing cost and $R - r + 1$ is the number of remaining rounds. The probability of accepting an inferior partitioning thus inversely depends on the current round r (i.e., how far the computation has progressed) and the change in expected processing cost. Otherwise, the partitioning is rejected, and the random intermediate bit is flipped back. In addition, Algorithm 9 keeps track of the best partitioning seen so far in `bestBoundaries` and `bestEPC`. After R rounds, the best partitioning seen is assembled from `bestBoundaries`.

Algorithm 9 has time complexity in $O(|L_v| + n^2 + n \cdot R)$ and space complexity in $O(n^2)$, if posting list sizes $|L_v : [t_i, t_j]|$ have been precomputed. Otherwise, its time complexity is in $O(R \cdot |L_v|)$ for computing the expected processing cost in time $O(|L_v|)$; its space complexity is in $O(n)$ for keeping `boundaries`.

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

3.8 Management of Time-Dependent Collection Statistics

Our retrieval model for time-travel keyword queries, described in Section 3.3, takes different time-dependent collection statistics into account. This section focuses on managing these time-dependent collection statistics in such a way that they can be retrieved efficiently at query-processing time.

Recall from Section 3.3 that our retrieval model considers the following time-dependent collection statistics:

- $N([t_b, t_e]) = |\mathbf{D}^{[t_b, t_e]}|$ as the number of document versions alive at any time in the query time-interval $[t_b, t_e]$.
- the document frequency $df(v, [t_b, t_e])$ as the number of document versions in $\mathbf{D}^{[t_b, t_e]}$ that contain the term v .

These time-dependent collection statistics are required at query-processing time to determine time-dependent idf-scores for the query terms. We now discuss how time-dependent document frequencies can be precomputed and managed so as to support their efficient look-up at query-processing time. Note that the same techniques can be applied for the collection size – we can simply introduce an artificial term that is present in every document version, so that its document frequency reflects the size of the collection.

For a given term v and a time interval $[t_b, t_e]$, the problem at hand is thus to identify the number of document versions alive at any point in $[t_b, t_e]$ that contain the term v . To solve this problem, we adopt the idea from Kanellakis et al. [KRVV96] that document versions alive at any point during $[t_b, t_e]$ can be disjointly separated into (i) document versions alive at time t_b and (ii) documents versions created at any time during $(t_b, t_e]$. Let $df(v, t_b)$ be the document frequency in document versions alive at t_b and $df_c(v, t)$ be the number of document versions created at time t that contain v , we can thus write

$$df(v, [t_b, t_e]) = df(v, t_b) + \sum_{t \in (t_b, t_e]} df_c(v, t) . \quad (3.37)$$

Given the list L_v of postings for term v , we can precompute the first summand for all $t \in T_v$ in time $O(|L_v| \cdot n)$ and space $O(n)$. Analogously, we could precompute $df_c(v, t)$ for all $t \in T_v$ in time $O(|L_v|)$ and space $O(n)$. At query-processing time, though, this would force us to compute the sum by scanning and aggregating values $df_c(v, t)$ for $t \in (t_b, t_e]$. Using prefix sums [HS86] this potentially expensive scanning and aggregation can be avoided. In detail, we define

$$df_b(v, t) = \sum_{t' < t} df_c(v, t') \quad (3.38)$$

as the number of document versions created before time t that contain v . These can be computed in time $O(|L_v| + n)$ and space $O(n)$. We can now rewrite Equation 3.37 as

$$df(v, [t_b, t_e]) = df(v, t_b) + df_b(v, t_e) - df_b(v, t_b) , \quad (3.39)$$

which can be computed efficiently at query-processing time using three look-ups, for instance, in B^+ -Trees that store precomputed values $df(v, t)$ and $df_b(v, t)$ for all $t \in T_v$ on disk.

3.9 FLUXCAPACITOR Prototype Implementation

We now describe FLUXCAPACITOR¹ – a prototype system that provides a full-fledged implementation of the techniques proposed in this work. Figure 3.8 depicts the overall system architecture and FLUXCAPACITOR’s main components, which we describe in more detail in the following.

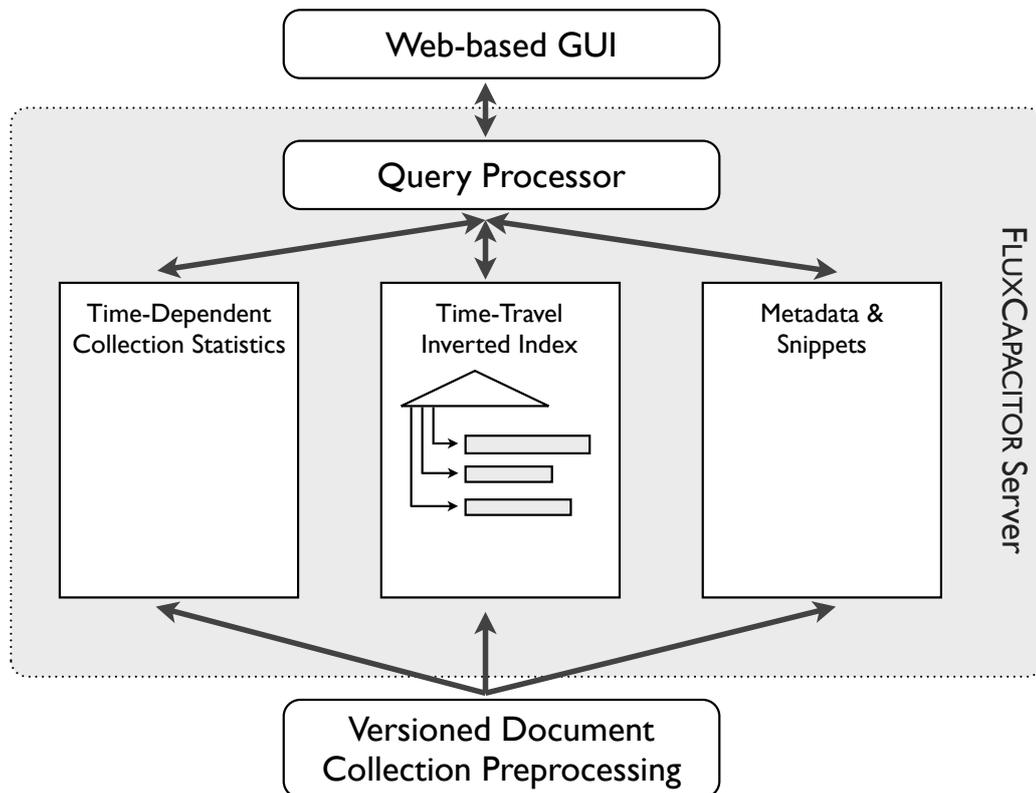


Figure 3.8: FLUXCAPACITOR’s System Architecture

¹ The prototype system’s name has its roots in the “Back to the Future” movie trilogy where the flux capacitor was the device that made time travel possible.

3.9.1 Web-Based GUI

Our **web-based GUI** is implemented using Google's Web Toolkit [GWT]. The toolkit allows users to implement the logic behind a web application, as well as required remote procedure calls (e.g., to web services) using Java. The Java code is then automatically translated into HTML and JavaScript that runs inside any web browser. Figure 3.9 shows two screenshots of the user interface that has the following key components: (1) *search box* for entering queries, (2) dynamically updated *result size estimate over time* for the entered query giving the user a hint on potentially interesting query time-points, (3) *timeline* by clicking on which the query time-point or time-interval is determined and the time-travel query is evaluated, and (4) *result presentation* including title, snippet, relevance score, as well as the publication date of the result document version.

When a time-travel query (e.g., *iraq war@June 26th, 2005*) is submitted via the GUI, it is sent to the **FLUXCAPACITOR Server**.

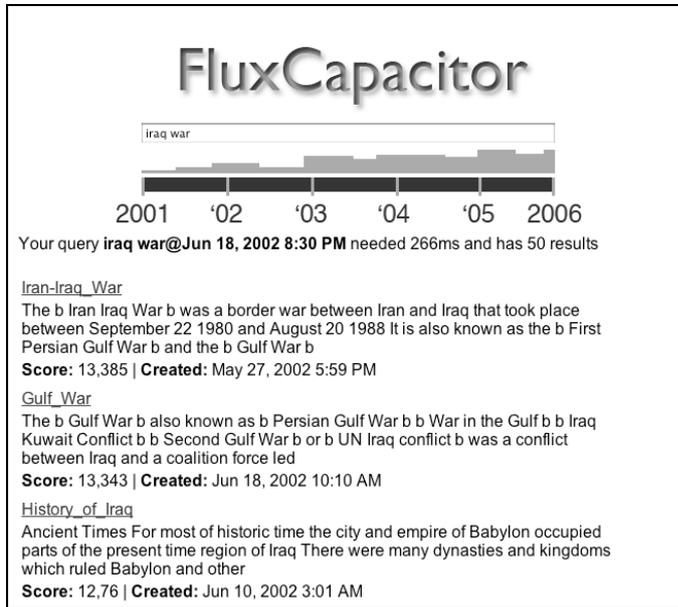
3.9.2 FLUXCAPACITOR Server

The FLUXCAPACITOR Server handles incoming time-travel queries. The server application is implemented as a Java servlet and runs inside an Apache Tomcat servlet engine.

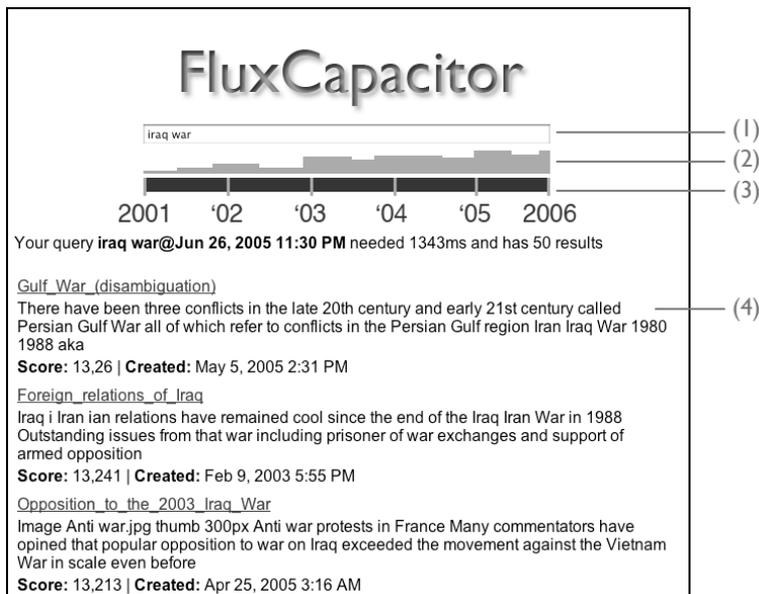
Query Processor

When a time-travel query is received by the FLUXCAPACITOR Server, the **Query Processor** component takes the following steps, before sending back the query result to the Web-based GUI.

- i) If needed, compute idf-scores for the query terms and the query time-interval of interest based on time-dependent collection statistics.
- ii) Selection of appropriate posting lists for each query term (as described in Section 3.5). Following that, the query is evaluated on the posting lists selected in the first step. We employ term-at-a-time query processing, as described in Chapter 2, to process Boolean queries. Keyword queries and phrase queries are processed using document-at-a-time query processing, i.e., by merging the identified posting lists.



(a) June 18th, 2002



(b) June 26th, 2005

Figure 3.9: FLUXCAPACITOR's Web-based GUI showing query results for the query *iraq war* and different query time-points

- iii) Enrichment of result document versions with metadata (e.g., the document version's title, URL, or a short text snippet describing its content).

Time-Dependent Collection Statistics

Our current implementation uses a simplified variant of the techniques discussed in Section 3.8. Thus, we do not use B⁺-Trees trees to store and look up values $df(v, t_b)$ and $df_b(v, t)$. Instead, we keep these values as arrays in flat files on disk. When we need to determine time-dependent document frequencies and collection sizes at query-processing time, the arrays for the required terms are fetched from disk into main memory and the required look-ups are performed.

Time-Travel Inverted Index

An instance of the time-travel inverted index, as computed in the versioned document collection preprocessing component detailed below, is stored on disk using flat files that contain the lexicon and the posting lists, respectively. As a bootstrapping step, the lexicon is read into main memory, so that posting lists can be selected efficiently. When a logical posting list is selected and thus needs to be retrieved from disk, two disk seeks are performed to retrieve the two underlying physical posting lists, which are then read sequentially and merged. Our implementation is versatile and includes different compression techniques such as 7-Bit and Elias γ - and δ -encoding described in Chapter 2. Details on which compression techniques we employ for different types of payloads are provided with our experimental evaluation in Section 3.10. When reading a posting list from disk, postings that are not required to process the query are filtered out, and coalesced postings are decoalesced using the in-memory document-version dictionary described below. Our temporal coalescing techniques and partitioning strategies thus remain transparent to the actual query processing. For the query processor it therefore always appears as if there was a posting list specifically built for the query time-interval of the current query.

Metadata & Snippets

As already mentioned in Section 3.6, we keep a document-version dictionary in main memory that records for each document the timestamps of its versions. Moreover, in our current implementation, the document-version dictionary re-

members for each document version where the corresponding metadata and snippet are stored on disk as an offset into a flat file.

3.9.3 Versioned Document Collection Preprocessing

The **Versioned Document Collection Preprocessing**, depicted in the bottom of Figure 3.8, includes the following steps:

- Removal of collection-specific markup, i.e., transformation of document versions into a collection-independent intermediate format.
- Indexing of document versions in our time-travel inverted index. This includes the tokenization of document versions, followed by an inversion step implemented using a multi-threaded external memory sort implementation. Finally, from the inverted document-version information, posting lists are determined, i.e., we execute for each term the pipeline consisting of temporal coalescing and partitioning.
- Precomputation of time-dependent collection statistics using the techniques discussed in Section 3.8 and leveraging the inverted document-version information computed when indexing the versioned document collection.
- Extraction and storage of metadata including the document version's title, a URL –if applicable–, and a short snippet describing its content.

The entire versioned document collection preprocessing is implemented using Java 1.6. All data, including time-dependent collection statistics, the time-travel inverted index, as well as metadata and snippets is kept in flat files on disk.

We wrap up this section on our prototype implementation FLUXCAPACITOR with some statistics about our code base. In total, our implementation comprises 19 packages, 125 classes that consist of nearly 10,000 lines Java code.

3.10 Experimental Evaluation

This section presents an experimental evaluation of our approach to time-travel text search. It examines the proposed temporal coalescing techniques and partitioning strategies with regard to their impact on *index size*, *result accuracy*, and *query-processing performance* on three real-world web archives.

3.10.1 Setup

System All experiments were run on Dell PowerEdge M610 servers, each of which has two Intel Xeon E5530 CPUs (resulting in a total of 8 CPU cores that run at 2.4 GHz), 48 GB of main memory, a large iSCSI-attached disk array, and runs Debian GNU/Linux (SMP Kernel 2.6.29.3.1) as an operating system. Experiments were conducted using the Java Hotspot 64-Bit Server Virtual Machine (build 11.2-b01) installed on our servers.

Implementation Details Our implementation is as described in Section 3.9. We neither applied stemming/lemmatization nor filtered out stopwords when indexing the datasets described below. For our low-level representation of different posting types we use gap encoding and 7-Bit encoding whenever applicable. In detail, for a posting $(d, [t_i, t_j], p)$ we employ 7-Bit encoding to store:

- the document identifier d .
- the valid-time interval represented as t_i and $(t_j - t_i)$.

Remember that nothing else needs to be stored for Boolean payloads. Scalar payloads, that capture tf-scores according to our adaptation of Okapi BM25 described in Section 3.3, are represented as a 64-Bit floating point number in our implementation and therefore consistently consume eight bytes. For positional payloads, we distinguish between postings that represent a single document version and coalesced postings, as described in Section 3.6, that contain information about word positions in subsequent versions of the same document. Word positions, in both kinds of postings, are stored using gap encoding and 7-Bit encoding. Bit signatures for the coalesced postings are stored explicitly, and we additionally enforce a byte-aligned posting representation by padding bit signatures if necessary. Thus, as an example, a bit signature representing the presence/absence of five word positions in four document versions consumes three bytes in our implementation because of this padding.

3.10.2 Datasets

Revision History of the English Wikipedia (WIKI) The revision history of the English Wikipedia (referred to as WIKI in the remainder), which is available for free download as a single XML file, serves as the first dataset in our experimental evaluation. This large dataset, whose uncompressed raw data amounts

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

European Archive Crawls of U.K. Governmental Websites (UKGOV) Our second dataset is based on a subset of the European Archive [EA] and contains weekly crawls of the eleven governmental websites from the U.K. listed in Table 3.1. The weekly crawls cover the years 2004 and 2005 and amount to about 2 TBytes of raw data. We filtered out documents not belonging to MIME-types `text/plain` and `text/html`, to obtain a dataset that totals 0.4 TBytes and is referred to as UKGOV in the following. This dataset includes 685,678 documents with 17,297,548 versions ($\mu = 25.23$ and $\sigma = 28.38$).

| Website | Description |
|----------------------------------------------------------------------------------|------------------------------------------------|
| www.army.mod.uk | The British Army Homepage |
| www.dfid.gov.uk | Department for International Development |
| www.doh.gov.uk | Department of Health |
| www.fco.gov.uk | Foreign & Commonwealth Office |
| www.mod.uk | Ministry of Defence |
| www.odpm.gov.uk | Office of the Deputy Prime Minister |
| www.pm.gov.uk | 10 Downing Street Website |
| www.raf.mod.uk | Royal Air Force |
| www.royal-navy.mod.uk | Royal Navy |
| www.sabre.mod.uk | Support for Britain's Reservists and Employers |
| www.the-hutton-inquiry.org.uk | The Hutton Inquiry |

Table 3.1: Websites contained in the UKGOV dataset

New York Times Annotated Corpus (NYT) The New York Times Annotated Corpus [NYT] is the third dataset used for our experimental evaluation. This dataset contains a total of 1,855,656 articles published in The New York Times between 1987 and 2007. In contrast to the other two datasets, NYT contains only one version per document. If we strictly followed our definition from Section 3.3, the valid-time interval of each document version would range from its

publication time until the current time now. In that case, though, time-travel text search degenerates, since both time-point queries and time-interval queries can be answered by simple range search on the publication times of documents. Therefore, we generate a more interesting and challenging variant of the dataset by assigning document versions a fixed-length valid-time interval that spans 30 days from the article’s publication time. We refer to this modified dataset as NYT-30 in the following. Note that this slight tinkering with the dataset has a counterpart in the real world, namely, if articles published in The New York Times were freely accessible only for thirty days after their publication. Time-travel text search on our dataset variant then searches only the portion of the corpus that was freely accessible at the user’s time of interest. When conducting our experimental evaluation, we also considered two additional variants of the dataset with valid-time intervals spanning seven and 90 days, respectively. Results for them are omitted, since they did not produce interesting findings beyond what we report for NYT-30.

For ease of reference, Table 3.2 summarizes and contrasts statistics about the three datasets used in our experiments.

| | WIKI | UKGOV | NYT-30 |
|--------------------------------------------|------------|------------|-----------|
| # Indexed terms | 4,791,840 | 5,261,386 | 1,227,226 |
| # Documents | 1,517,524 | 685,678 | 1,855,656 |
| # Document versions | 15,079,829 | 17,297,548 | 1,855,656 |
| # Versions per document (μ) | 9.94 | 25.23 | 1.00 |
| # Versions per document (σ) | 46.08 | 28.38 | 0.00 |
| Document length in characters (μ) | 8,751.47 | 8,845.93 | 3,211.45 |
| Document length in characters (σ) | 17,544.69 | 13,772.89 | 3,408.43 |
| Version lifespan in days (μ) | 23.68 | 7.26 | 30.00 |
| Version lifespan in days (σ) | 73.78 | 9.50 | 0.00 |

Table 3.2: Dataset statistics (with mean μ and standard deviation σ)

Query Workloads Since we want to examine our approach to time-travel text search on *Boolean queries*, *keyword queries*, and *phrase queries*, we need realistic query workloads for each of these query types. To build such query workloads, we leverage the query logs that were temporarily made available by AOL Research in 2006. For each dataset, we compile two query workloads by extracting

frequent queries from the AOL query log that yielded a result click on a domain having one of the relevant suffixes listed in Table 3.3. The first query workload per dataset contains the 150 queries that are most frequent – these queries serve as Boolean queries and keyword queries. The second query workload contains the 150 queries that consist of at least two query terms and correspond to titles of Wikipedia articles. The rationale behind constructing the second workload in this way is that the identified queries would often correspond to entity names and therefore constitute sensible phrase queries. We thus obtain six query workloads. Figure 3.10 shows excerpts of the query workloads – the query workloads are given in their entirety in Appendix A.

| Dataset | Domain Suffixes |
|---------|------------------------------------------------------------------|
| WIKI | en.wikipedia.org |
| UKGOV | gov.uk |
| NYT | nyt.com, nytimes.com, wsj.com, ft.com, washingtonpost.com |

Table 3.3: Relevant domain suffixes per dataset

We derive 50 time-travel queries from each of the queries extracted from the AOL query log by enriching them with randomly chosen time points and time intervals. Thereby, we consider five granularities for the query time-intervals, namely, (a) *millisecond* (corresponding to time-point queries in our setup), (b) *a single day*, (c) *seven days*, (d) *30 days*, and (e) *365 days*, where (c)–(e) roughly correspond to weeks, months, and years, respectively. Given this rough correspondence, we simply refer to the five query time-interval granularities as (a) *MS*, (b) *D*, (c) *W*, (d) *M*, and (e) *Y*. For each of them, ten time-travel queries are generated by randomly choosing the begin boundary of a query time-interval of the respective granularity, such that the time interval falls entirely into the time window that is covered by the considered document collection. As a result, for each dataset we thus obtain one time-travel Boolean/keyword query workload and one time-travel phrase query workload, each containing a total of 7,500 time-travel queries.

When processing queries from our workloads in the following experiments, we apply the following semantics. For Boolean queries all terms are mandatory, i.e., our query results only contain document versions that include all query terms. Keyword queries are evaluated using disjunctive query semantics, i.e., a reported query result may not contain all query terms. Phrase queries, finally,

| Dataset | Boolean/Keyword Queries | Phrase Queries |
|---------|--------------------------------------------------------------------------------|--------------------------------------------------------------------|
| WIKI | french revolution, industrial revolution, kkk, columbine | alexander the great, gospel of judas, cinco de mayo |
| UKGOV | inheritance tax, statistics, toyota city japan, mi6, doh, dsa | queen elizabeth ii, city of liverpool, william the conqueror |
| NYT | flowers, political cartoons, sudoku, cold war, sports, weather, msnbc, laptops | seven wonders of the world, william kennedy smith, the war in iraq |

Figure 3.10: Excerpts from query workloads

are evaluated in a strict sense, i.e., only document versions containing the exact query phrase can become part of the query result.

Indexes Built On each of our datasets we built indexes with Boolean, scalar, and positional payloads using combinations of the suitable temporal coalescing technique –if applicable to the dataset– and different partitioning strategies. In detail, we consider the following partitioning strategies and parameter choices:

- Performance-Optimal Partitioning (P_{opt}).
- Space-Optimal Partitioning (S_{opt}).
- Performance-Guarantee Partitioning (PG) with

$$\gamma \in \{1.10, 1.25, 1.50, 2.00, 2.50, 3.00\}$$

using the optimal Algorithm 6.

- Space-Bound Partitioning (SB) with

$$\kappa \in \{3.00, 2.50, 2.00, 1.50, 1.25, 1.10\}$$

using the approximate Algorithm 9 running the underlying simulated annealing for $R = 10^5$ rounds.

When applying the partitioning strategies, we round document timestamps to day granularity. The valid-time intervals of postings stored in the created indexes, though, are kept at the millisecond granularity.

For brevity, we discuss the impact of temporal coalescing and partitioning strategies on index size and query-processing performance, respectively, in isolation. Thus, the impact of temporal coalescing is discussed assuming that the

space-optimal partitioning strategy S_{opt} is employed. Analogously, when discussing the impact of different partitioning strategies, we choose a fixed parameter for the appropriate coalescing technique. When conducting our experiments, we computed indexes exhaustively, considering all combinations of temporal coalescing and temporal partitioning strategies. The results obtained on these indexes confirm our findings discussed below, but do not provide additional insights, and are therefore omitted.

3.10.3 Index Size

This first part of our experimental evaluation examines how temporal coalescing techniques and partitioning strategies impact index size. When reporting index sizes, we give both the total number of postings in millions that is kept in the index (**#P (M)**), as an implementation-independent measure, as well as the total number of MBytes that the index built using our implementation consumes on hard disk (**MBytes**).

Boolean Payloads

Temporal coalescing for Boolean payloads is parameter-free, i.e., it is only a question whether temporal coalescing is used or not. Table 3.4 lists the index sizes obtained for the WIKI and UKGOV dataset when not employing temporal coalescing and those obtained when employing temporal coalescing for Boolean payloads. Note that the NYT-30 dataset is not present in Table 3.4, since there is no point in applying temporal coalescing to it, given that there is only a single version per document.

Temporal coalescing for Boolean payloads is highly effective, as the figures given in Table 3.4 reveal. For the WIKI dataset, temporal coalescing reduces the number of postings with Boolean payload to less than 5% of the number of postings in the non-coalesced index. Although slightly less effective on the UKGOV dataset, temporal coalescing still reduces the number of postings considerably to less than 9% of the total number of postings in the non-coalesced index. Actual index sizes measured for our implementation reflect these reductions by more than one order of magnitude that temporal coalescing achieves for indexes with Boolean payloads.

Table 3.5 shows index sizes obtained for different *partitioning strategies*. For the WIKI and UKGOV dataset the index sizes were determined using temporal coa-

| | WIKI | | UKGOV | |
|----------------------|---------|--------|---------|--------|
| | # P (M) | MBytes | # P (M) | MBytes |
| Non-Coalesced | 6,928 | 89,254 | 5,355 | 85,218 |
| Coalesced | 314 | 5,044 | 392 | 6,895 |

Table 3.4: Impact of temporal coalescing on index size for Boolean payloads on WIKI and UKGOV

| | WIKI | | UKGOV | | NYT-30 | |
|-----------------------|---------|-----------|---------|-----------|---------|---------|
| | # P (M) | MBytes | # P (M) | MBytes | # P (M) | MBytes |
| P_{opt} | 60,911 | 1,054,611 | 69,118 | 1,337,800 | 13,606 | 191,554 |
| PG($\gamma = 1.10$) | 3,158 | 54,184 | 3,256 | 62,344 | 4,658 | 65,820 |
| PG($\gamma = 1.25$) | 1,496 | 25,462 | 1,739 | 33,019 | 2,517 | 35,647 |
| PG($\gamma = 1.50$) | 917 | 15,449 | 1,135 | 21,349 | 1,541 | 21,862 |
| PG($\gamma = 2.00$) | 620 | 10,315 | 791 | 14,678 | 1,032 | 14,657 |
| PG($\gamma = 2.50$) | 522 | 8,616 | 669 | 12,302 | 861 | 12,241 |
| PG($\gamma = 3.00$) | 471 | 7,745 | 608 | 11,114 | 770 | 10,955 |
| SB($\kappa = 3.00$) | 929 | 15,663 | 1,160 | 21,767 | 1,449 | 20,553 |
| SB($\kappa = 2.50$) | 776 | 13,025 | 971 | 18,115 | 1,211 | 17,175 |
| SB($\kappa = 2.00$) | 624 | 10,386 | 782 | 14,455 | 971 | 13,778 |
| SB($\kappa = 1.50$) | 470 | 7,720 | 589 | 10,725 | 730 | 10,356 |
| SB($\kappa = 1.25$) | 393 | 6,384 | 474 | 8,494 | 609 | 8,635 |
| SB($\kappa = 1.10$) | 346 | 5,598 | 413 | 7,301 | 536 | 7,597 |
| S_{opt} | 314 | 5,044 | 392 | 6,895 | 488 | 6,831 |

Table 3.5: Impact of partitioning strategies on index size for Boolean payloads on WIKI, UKGOV, and NYT-30

lescoring for Boolean payloads. For NYT-30 the application of temporal coalescing is not sensible, as argued above. The figures clearly show that the differences in index size between the performance-optimal partitioning strategy P_{opt} and its space-optimal counterpart S_{opt} are vast. For the WIKI dataset, for instance, the index produced by P_{opt} is nearly 193 times larger than the index produced by its space-optimal counterpart. Such vast differences between the two extreme partitioning strategies underline once more the need for partitioning strategies that allow to explore the spectrum between the two extremes. The figures given in Table 3.5 show that our PG and SB partitioning strategies cater to this need. For the PG partitioning strategy that comes with a performance guarantee for time-point queries, we already see that the index produced for the parameter value $\gamma = 1.10$, which is expected to give close-to-optimal performance, is smaller by more than one order of magnitude than the index produced by P_{opt} .

Scalar Payloads

Temporal coalescing for scalar payloads is tunable by the parameter ϵ that controls the maximum acceptable relative approximation error. We consider the six parameter choices

$$\epsilon \in \{0.00, 0.01, 0.05, 0.10, 0.25, 0.50\}$$

and also study the case where temporal coalescing is not used. Table 3.6 gives the resulting index sizes for the WIKI and UKGOV dataset, when employing the space-optimal partitioning strategy S_{opt} .

As can be seen from the figures, on both datasets index size is already reduced substantially when choosing $\epsilon = 0.01$. Further increases of ϵ result in noticeable but less substantial reductions of index size. As a concrete figure, the index sizes in MBytes obtained for $\epsilon = 0.10$ (our parameter choice for investigating partitioning strategies below) amount to about 53% of their non-coalesced counterparts on both datasets.

For $\epsilon = 0.00$ our temporal coalescing for scalar payloads corresponds to an application of traditional temporal coalescing, as proposed for temporal databases. Interestingly, for the WIKI dataset, the technique accomplishes hardly any reduction in index size for this parameter choice. On the UKGOV dataset, in contrast, temporal coalescing already achieves a substantial reduction in index size when setting $\epsilon = 0.00$. The reason for this lies in the different characteristics of

| | WIKI | | UKGOV | |
|----------------------|---------|---------|---------|---------|
| | # P (M) | MBytes | # P (M) | MBytes |
| Non-Coalesced | 6,928 | 144,685 | 5,355 | 128,387 |
| $\epsilon = 0.00$ | 6,632 | 138,681 | 3,277 | 78,939 |
| $\epsilon = 0.01$ | 3,725 | 79,230 | 2,846 | 68,691 |
| $\epsilon = 0.05$ | 3,592 | 76,511 | 2,828 | 68,263 |
| $\epsilon = 0.10$ | 3,570 | 76,055 | 2,815 | 67,960 |
| $\epsilon = 0.25$ | 3,554 | 75,731 | 2,781 | 67,134 |
| $\epsilon = 0.50$ | 3,551 | 75,672 | 2,775 | 66,999 |

Table 3.6: Impact of temporal coalescing on index size for scalar payloads on WIKI and UKGOV

| | WIKI | | UKGOV | | NYT-30 | |
|---------------------|---------|-----------|---------|-----------|---------|---------|
| | # P (M) | MBytes | # P (M) | MBytes | # P (M) | MBytes |
| P_{opt} | 68,446 | 1,570,510 | 69,118 | 1,662,118 | 13,606 | 300,407 |
| $PG(\gamma = 1.10)$ | 28,723 | 658,614 | 19,520 | 509,131 | 4,658 | 103,087 |
| $PG(\gamma = 1.25)$ | 15,475 | 351,927 | 11,390 | 293,795 | 2,517 | 55,782 |
| $PG(\gamma = 1.50)$ | 9,995 | 225,000 | 7,699 | 196,349 | 1,541 | 34,188 |
| $PG(\gamma = 2.00)$ | 6,944 | 154,251 | 5,515 | 138,711 | 1,032 | 22,911 |
| $PG(\gamma = 2.50)$ | 5,876 | 129,515 | 4,669 | 116,540 | 861 | 19,126 |
| $PG(\gamma = 3.00)$ | 5,327 | 116,772 | 4,245 | 105,392 | 770 | 17,117 |
| $SB(\kappa = 3.00)$ | 10,593 | 237,562 | 8,430 | 217,160 | 1,449 | 32,147 |
| $SB(\kappa = 2.50)$ | 8,859 | 197,761 | 7,037 | 180,165 | 1,211 | 26,861 |
| $SB(\kappa = 2.00)$ | 7,113 | 157,628 | 5,637 | 142,958 | 971 | 21,548 |
| $SB(\kappa = 1.50)$ | 5,351 | 117,099 | 4,232 | 105,592 | 730 | 16,199 |
| $SB(\kappa = 1.25)$ | 4,464 | 96,685 | 3,529 | 86,890 | 609 | 13,510 |
| $SB(\kappa = 1.10)$ | 3,930 | 84,371 | 3,108 | 75,675 | 537 | 11,891 |
| S_{opt} | 3,570 | 76,055 | 2,815 | 67,960 | 488 | 10,734 |

Table 3.7: Impact of partitioning strategies on index size for scalar payloads on WIKI, UKGOV, and NYT-30

the two datasets. Subsequent snapshots of a web page in the UKGOV dataset may be completely identical, thus resulting in identical Okapi BM25 tf-scores that can be coalesced in the traditional way. Subsequent revisions of a Wikipedia article, in contrast, typically differ at least in their document length, thus resulting in slightly different Okapi BM25 tf-scores that cannot be coalesced in the traditional manner. This shows that allowing for small upper-bounded errors, as our technique does, is crucial to reduce index size on general datasets. Giving additional leeway to temporal coalescing and allowing for small upper-bounded errors, though, may distort query results, as we explained above. Therefore, we study the impact of temporal coalescing for scalar payloads on the accuracy of query results in another experiment below.

Table 3.7 gives index sizes obtained for different *partitioning strategies*. For the WIKI and UKGOV dataset, we used the parameter setting $\epsilon = 0.10$ for temporal coalescing. For the reason explained above, indexes for NYT-30 were built without employing temporal coalescing. Again, observe the vast difference in size between the indexes built using P_{opt} and S_{opt} . Whereas, for the WIKI dataset, the former produces an index that consumes more than 1.5 TBytes, the index produced by the latter only consumes little more than 76 GBytes. The PG and SB partitioning strategies proposed in this work yield indexes whose sizes lie in-between these two extremes. As can be seen from the figures, the PG partitioning strategies makes effective use of additional space. Thus, the index constructed for the parameter choice, $\gamma = 1.50$, which guarantees a degradation of performance for time-point queries by at most 50%, consumes only 225 GBytes – less than 15% the space required by P_{opt} .

Positional Payloads

Temporal coalescing for positional payloads is tunable by the parameter η that controls the worst-case overhead factor. We consider the four parameter choices

$$\eta \in \{1.00, 1.50, 2.50, 5.00\}$$

and the case where temporal coalescing is not employed. When using the space-optimal partitioning technique S_{opt} , we obtained the index sizes given in Table 3.8.

As the reported index sizes demonstrate, temporal coalescing for positional payloads is highly effective. When choosing $\eta = 1.00$, which means that no

overhead in terms of bytes read is accepted, the number of MBytes consumed by the indexes on disk is reduced by about 20% on WIKI and by almost 50% on UKGOV. The significant difference between the achieved reductions, again, is a result of the two datasets' different characteristics that we alluded to earlier. Index sizes continue to shrink as we increase the parameter η further. For $\eta = 1.50$ (our parameter choice for investigating partitioning strategies), as two concrete figures, the indexes constructed for WIKI and UKGOV using the space-optimal partitioning strategy consume about 38 GBytes and 20 GBytes, respectively – clearly substantial savings when compared to the 120 GBytes and 118 GBytes consumed by their non-coalesced counterparts.

Index sizes obtained when using the different *partitioning strategies* are given in Table 3.9. For the WIKI and UKGOV dataset we employed temporal coalescing for positional payloads using the parameter choice $\eta = 1.50$. Indexes for NYT were again built without employing temporal coalescing.

The figures shown support our earlier observations that the difference in size between the indexes built using S_{opt} and P_{opt} is vast, but our PG and SB partition strategies allow for fine-tuning the index between these two extremes. Further, note that for NYT-30, that does not profit from temporal coalescing, index sizes measured in terms of their total number of postings are identical between different posting payloads (as reported in Table 3.5, Table 3.7, and Table 3.9). This is expected given that partitioning strategies only take into account postings' valid-time intervals but not their payloads. When comparing our indexes built on the NYT-30 dataset with scalar and positional payloads, we observe that the former consistently consume more space than the latter. This may seem unintuitive first, given that indexes with positional payloads store more detailed information about term occurrences in documents. However, note that this is an artifact of our implementation, which eagerly compresses positional payloads, but represents scalar payloads using a fixed number of bytes.

Finally, for the WIKI and UKGOV dataset, when comparing the sizes of indexes with different payloads built using the same partitioning strategy, we see the synergetic effect between temporal coalescing and partition strategies that we mentioned above. Consider, as a concrete example, the indexes with scalar payloads and positional payloads, respectively, built using PG($\gamma = 1.10$) for the WIKI dataset. Since temporal coalescing for positional payloads with the parameter choice $\eta = 1.50$ results in only $1,297 \times 10^6$ distinct postings (as given in Table 3.8) as opposed to $3,570 \times 10^6$ distinct postings obtained for scalar pay-

| | WIKI | | UKGOV | |
|----------------------|---------|---------|---------|---------|
| | # P (M) | MBytes | # P (M) | MBytes |
| Non-Coalesced | 6,928 | 119,755 | 5,355 | 117,940 |
| $\eta = 1.00$ | 5,284 | 110,407 | 2,110 | 62,131 |
| $\eta = 1.50$ | 1,297 | 37,870 | 565 | 19,995 |
| $\eta = 2.50$ | 717 | 29,602 | 413 | 16,508 |
| $\eta = 5.00$ | 607 | 28,789 | 379 | 15,720 |

Table 3.8: Impact of temporal coalescing on index size for positional payloads on WIKI and UKGOV

| | WIKI | | UKGOV | | NYT-30 | |
|---------------------|---------|-----------|---------|-----------|---------|---------|
| | # P (M) | MBytes | # P (M) | MBytes | # P (M) | MBytes |
| P_{opt} | 59,060 | 1,504,915 | 69,118 | 1,598,228 | 13,606 | 249,903 |
| $PG(\gamma = 1.10)$ | 11,625 | 300,996 | 4,785 | 151,728 | 4,658 | 85,346 |
| $PG(\gamma = 1.25)$ | 5,866 | 154,415 | 2,509 | 80,231 | 2,517 | 46,221 |
| $PG(\gamma = 1.50)$ | 3,686 | 98,923 | 1,622 | 52,400 | 1,541 | 28,301 |
| $PG(\gamma = 2.00)$ | 2,536 | 69,633 | 1,137 | 37,166 | 1,032 | 18,959 |
| $PG(\gamma = 2.50)$ | 2,148 | 59,746 | 965 | 31,793 | 861 | 15,825 |
| $PG(\gamma = 3.00)$ | 1,947 | 54,622 | 876 | 29,005 | 770 | 14,164 |
| $SB(\kappa = 3.00)$ | 3,825 | 102,348 | 1,709 | 54,990 | 1,449 | 26,642 |
| $SB(\kappa = 2.50)$ | 3,205 | 86,587 | 1,428 | 46,259 | 1,210 | 22,258 |
| $SB(\kappa = 2.00)$ | 2,579 | 70,678 | 1,149 | 37,535 | 971 | 17,852 |
| $SB(\kappa = 1.50)$ | 1,943 | 54,478 | 866 | 28,710 | 730 | 13,416 |
| $SB(\kappa = 1.25)$ | 1,623 | 46,274 | 724 | 24,222 | 609 | 11,186 |
| $SB(\kappa = 1.10)$ | 1,430 | 41,342 | 637 | 21,543 | 537 | 9,841 |
| S_{opt} | 1,297 | 37,870 | 565 | 19,995 | 488 | 8,867 |

Table 3.9: Impact of partitioning strategies on index size for positional payloads on WIKI, UKGOV, and NYT-30

loads with the parameter choice $\epsilon = 0.10$, the indexes with positional payloads determined by different partitioning strategies contain generally fewer postings than their counterparts with scalar payloads, as can be seen from Table 3.7 and Table 3.9.

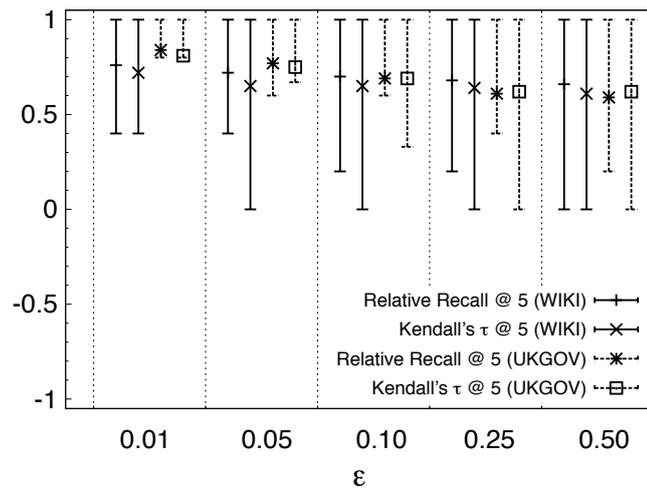
3.10.4 Result Accuracy

Our temporal coalescing technique for scalar payloads introduces an approximation to query results, as we discussed in Section 3.6. Temporal coalescing for scalar payloads reduces index sizes substantially, as our above experiments demonstrate. For the parameter choice $\epsilon = 0.10$, as an example, we observed a reduction in index size by about 50%. This reduction in index size, though, is only useful if query results computed on the compact coalesced index are close to the original query results. Or to put it differently, if temporal coalescing for scalar payloads does not heavily distort query results. In our second experiment, we therefore systematically evaluate the important aspect of result accuracy.

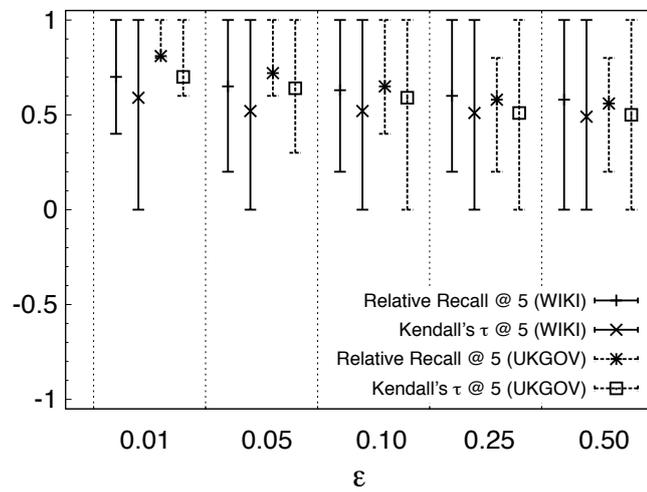
To this end, we again employ the time-travel keyword-query workloads for the WIKI and UKGOV dataset that were described above. When studying the impact of temporal coalescing on the result accuracy of time-travel keyword queries, we explore the following three dimensions: (i) the choice of ϵ controlling the relative error that temporal coalescing is permitted to make, (ii) the cut-off level k , (iii) the granularity of the query time-interval. We consider parameter choices $\epsilon \in \{0.01, 0.05, 0.10, 0.25, 0.50\}$, thus omitting the assignment $\epsilon = 0.00$ for which query results are by definition identical to the original query results. For the cut-off level we consider choices $k \in \{5, 10, 25, 100\}$. For brevity, we only report results obtained for query time-intervals corresponding to milliseconds (MS), weeks (W), and years (Y).

Figure 3.11–3.14 give mean result accuracies, measured using relative recall and Kendall’s τ as described in Chapter 2, along with their 25%-percentile and 75%-percentile for increasing cut-off levels k . The figures shown support the following observations.

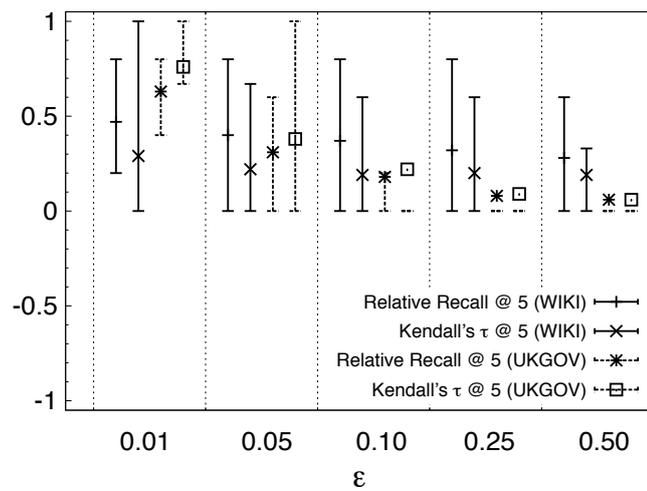
Result accuracy improves for all query time-interval granularities when we increase the cut-off level k , as can be seen from comparing corresponding plots for the same query time-interval granularity across Figure 3.11–3.14. As a concrete figure, on the UKGOV dataset for the parameter choice $\epsilon = 0.10$, the mean relative recall for queries having a millisecond-granularity query time-interval



(a) Millisecond (MS)

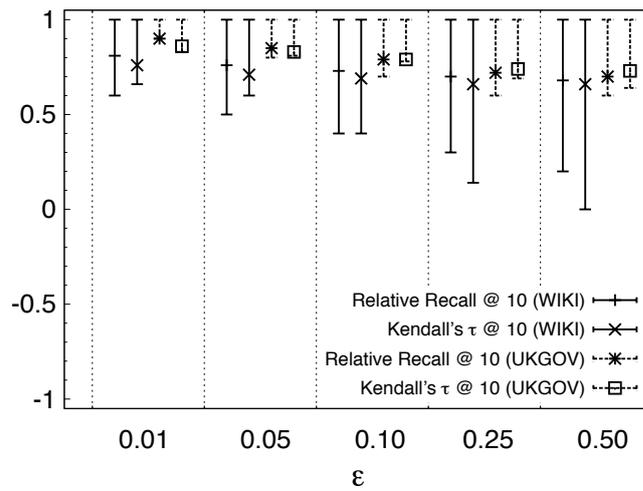


(b) Week (W)

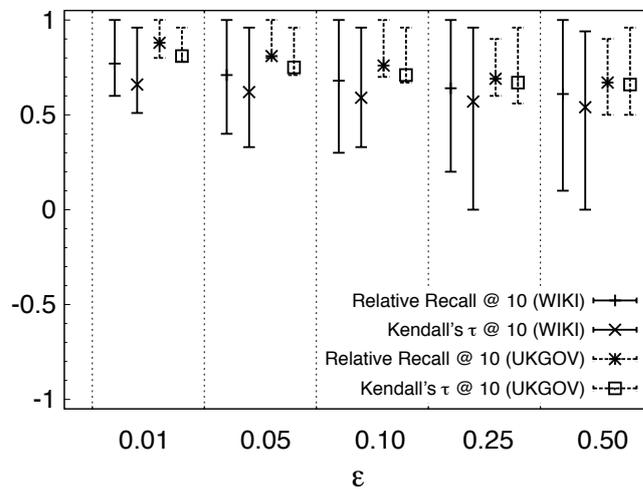


(c) Year (Y)

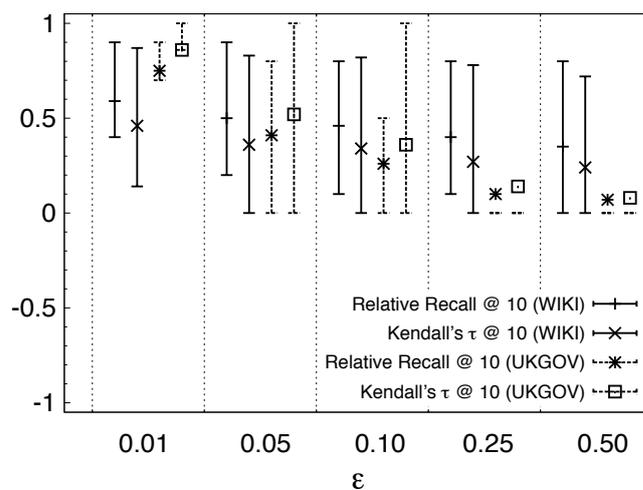
Figure 3.11: Result accuracy at cut-off level $k = 5$ on WIKI and UKGOV



(a) Millisecond (MS)

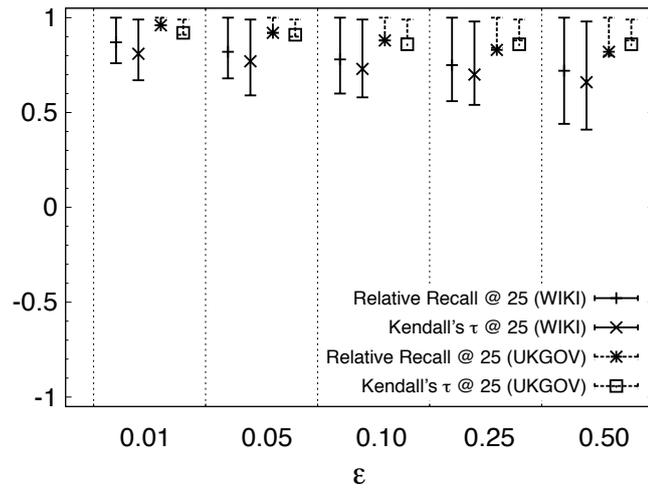


(b) Week (W)

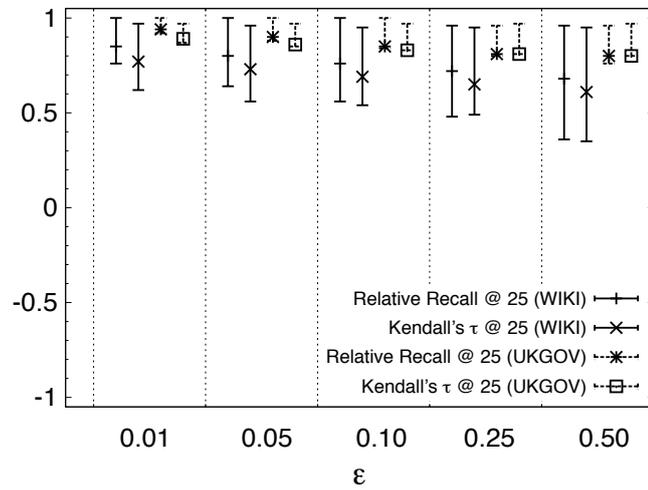


(c) Year (Y)

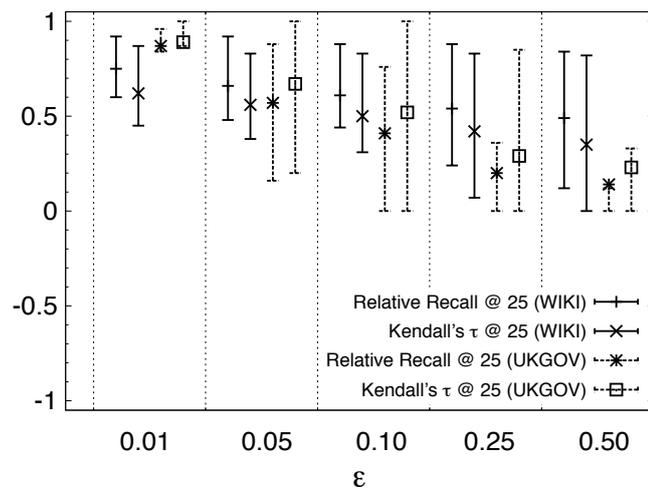
Figure 3.12: Result accuracy at cut-off level $k = 10$ on WIKI and UKGOV



(a) Millisecond (MS)

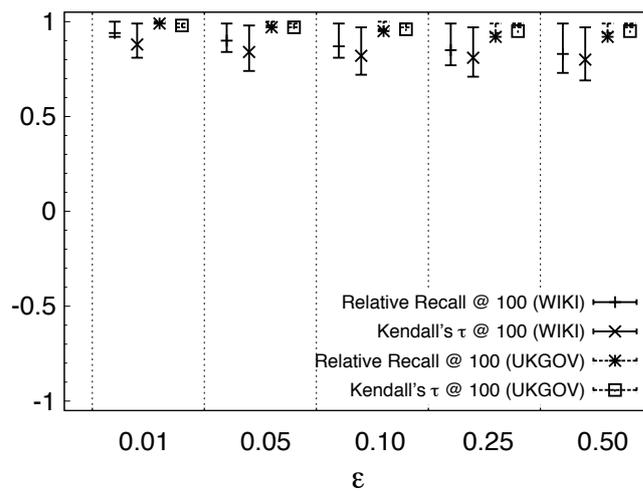


(b) Week (W)

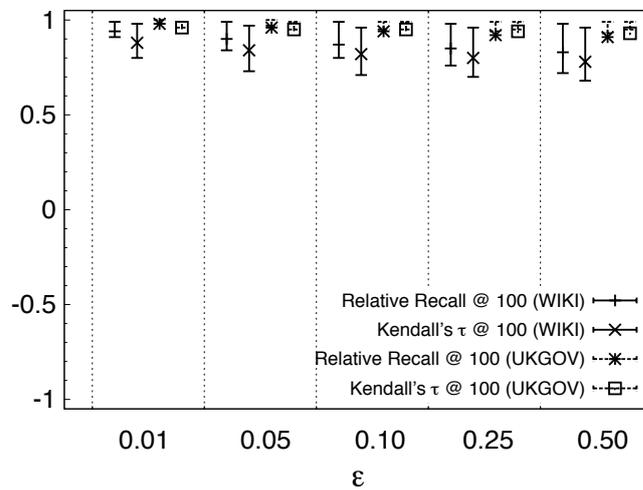


(c) Year (Y)

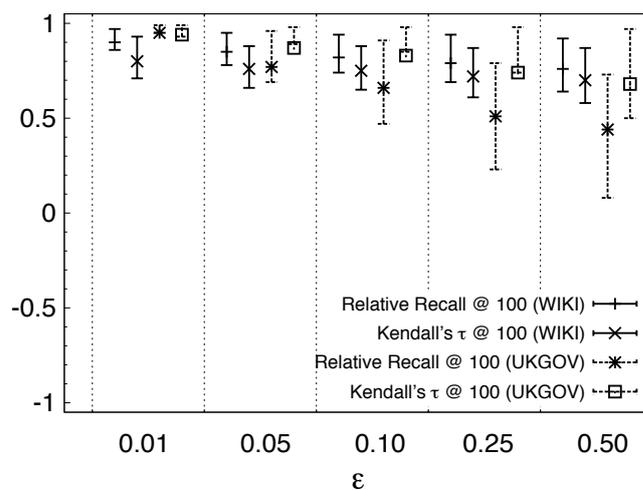
Figure 3.13: Result accuracy at cut-off level $k = 25$ on WIKI and UKGOV



(a) Millisecond (MS)



(b) Week (W)



(c) Year (Y)

Figure 3.14: Result accuracy at cut-off level $k = 100$ on WIKI and UKGOV

is 0.69 when looking at only the top-5 but 0.88 for top-25 query results.

As we increase the value of ϵ and thus give more leeway to temporal coalescing, result accuracies degrade gracefully. Consider, for instance, Figure 3.12, where the mean relative recall for queries on the WIKI dataset at cut-off level $k = 10$ with a millisecond-granularity query time-interval drops from 0.81 to 0.73 as we increase the value of ϵ from 0.01 to 0.1 – on average we thus retrieve only seven instead of eight original query results.

When increasing the granularity of the query time-interval, result accuracy is reduced, as can be seen from comparing the three plots shown in each of Figure 3.11–3.14. For instance, in Figure 3.11, on the WIKI dataset for $\epsilon = 0.10$ and cut-off level $k = 5$, we observe a mean relative recall of 0.72 for queries having a millisecond-granularity query-time interval; in the same setup, the mean relative recall for queries having a year-granularity time-interval is significantly lower at 0.37. The corresponding mean Kendall’s τ values exhibit an even larger decrease from 0.65 to 0.19, meaning that the fewer original results retrieved for queries having a year-granularity time-interval also deviate more from their original order.

Comparing result accuracies across the two datasets, we observe that the result accuracy for queries having a query time-interval granularity corresponding to milliseconds or weeks is comparable or better on the UKGOV dataset. For queries with a year-granularity time-interval, though, relative recall is consistently worse on the UKGOV dataset than on the WIKI dataset. This is an effect of the on average larger number of versions per document in the UKGOV dataset, as given in Table 3.2, and the way how our result accuracies are determined. Note that we compute result accuracies in a pessimistic manner, treating different versions of the same document as different query results. Therefore, even if two document versions are almost-identical and look the same to the user, result accuracy would reduce, if one is retrieved instead of the other.

For the parameter choice $\epsilon = 0.10$, that we use to study the impact of our partitioning strategies on index size and query-processing performance, the obtained result accuracies mean the following for query results: When looking at the top-10 results for a time-point keyword query, on average we see seven of the original query results on both datasets. Furthermore, as the corresponding 25%-percentiles reveal for 75% of our queries we see at least four and seven original query results on the WIKI dataset and UKGOV dataset, respectively. When looking at week-granularity query time-intervals, on average we still retrieve

six original query results on WIKI and seven original query results on UKGOV. For queries with a query time-interval that corresponds to a year, finally, on average four and two of the original query results are in the result computed on the compact coalesced index on WIKI and UKGOV.

Note that in our experiments, the left-out query time-interval granularities corresponding to days and months also exhibited behavior in support of the above observations.

3.10.5 Query-Processing Performance

Query-processing performance is the third and final aspect of interest that we examine experimentally. We employ two measures to quantify the query-processing performance achieved by the different indexes.

Expected processing cost (**EPC**), as the first measure, is the expected number of postings that needs to be read from the index for a single query term, picked at uniform random from the vocabulary, and a query time-point, picked at uniform random within the time window covered by the document collection. Hence, the expected processing cost is a query-independent and implementation-independent measure that inversely reflects query-processing performance of time-point queries.

We employ query-processing wallclock-times as a second measure of query-processing performance. Query-processing wallclock-times are reported in milliseconds (**ms**) and were measured on warm caches using only a single CPU core. In detail, every time-travel query was executed four times in a row: once to warm caches and, subsequently, three times to obtain a more stable measurement of the query-processing time.

In the following, we report our findings regarding processing performance of Boolean queries, keyword queries, and phrase queries. For each query type, we examine our five different query time-interval granularities separately. Again, as for the index size above, the impact of temporal coalescing and partitioning strategies is studied in isolation.

Boolean Queries

Boolean queries are evaluated on the indexes with Boolean payloads whose sizes are given in Table 3.4 and Table 3.5.

Query-processing performance figures obtained when employing alternatively not employing *temporal coalescing* for Boolean payloads and using the S_{opt} partitioning strategy are given for the WIKI and UKGOV dataset in Table 3.10.

| | WIKI | | | | | |
|----------------------|----------|---------|--------|--------|--------|--------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| Non-Coalesced | 1,445.80 | 483.07 | 475.53 | 488.13 | 537.00 | 847.02 |
| Coalesced | 65.60 | 34.64 | 34.67 | 35.37 | 36.13 | 43.86 |

| | UKGOV | | | | | |
|----------------------|----------|----------|----------|----------|----------|----------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| Non-Coalesced | 1,017.88 | 2,076.07 | 2,081.16 | 2,168.14 | 2,363.04 | 5,438.67 |
| Coalesced | 74.58 | 406.75 | 414.22 | 478.69 | 675.11 | 4,141.95 |

Table 3.10: Impact of temporal coalescing on the processing performance of Boolean queries on WIKI and UKGOV

As the figures reveal, temporal coalescing leads to considerable improvements in query-processing performance across both datasets and all query time-interval granularities. On the WIKI dataset, time-point queries are processed on average in less than 35 ms when employing temporal coalescing, which is an improvement by a factor of 13 in comparison to the query-processing times observed on the non-coalesced index. On the UKGOV dataset, temporal coalescing improves average wallclock times for time-point queries by a factor of 5. When looking at other query time-interval granularities, we observe that the two datasets exhibit different behavior. On the WIKI dataset, as we consider larger query time-interval granularities, temporal coalescing is increasingly effective. Thus, queries with a year-granularity time interval are evaluated in about 44 ms on the coalesced index, which corresponds to an improvement by a factor of 19 over the 847 ms observed on the non-coalesced index. In contrast, on the UKGOV dataset, the impact of temporal coalescing on query-processing performance reduces as we consider larger query time-interval granularities. When looking at queries with a year-granularity query time-interval, we see that the average query-processing wallclock time reduces from 5,439 ms to 4,142 s, which corresponds to an improvement by only a factor of 1.3. To see why this is natural, recall that the UKGOV dataset covers only two years, whereas the WIKI dataset covers about five years. Therefore, a query with a year-granularity

query time-interval is less selective on UKGOV than on WIKI. When processing such a query, on WIKI many coalesced postings can be filtered out immediately, whereas on UKGOV most of them need to be considered and be decoalesed using the in-memory document-version dictionary. The in general larger magnitude of wallclock times observed on the UKGOV dataset is an effect of the in general lower selectivity of our query workloads for the UKGOV dataset that contain many stopwords (e.g., of, the, in) or terms that are generally frequent in this dataset (e.g., uk, royal, doh).

| | WIKI | | | | | |
|-----------------------|-------|---------|--------|--------|--------|--------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| P_{opt} | 9.63 | 15.33 | 16.24 | 17.46 | 22.69 | 84.86 |
| PG($\gamma = 1.10$) | 10.28 | 15.59 | 16.65 | 16.49 | 17.36 | 31.46 |
| PG($\gamma = 1.25$) | 11.09 | 15.79 | 16.87 | 16.62 | 17.69 | 29.18 |
| PG($\gamma = 1.50$) | 12.33 | 15.81 | 17.17 | 16.88 | 18.05 | 28.60 |
| PG($\gamma = 2.00$) | 14.47 | 16.88 | 17.62 | 17.76 | 18.72 | 29.51 |
| PG($\gamma = 2.50$) | 16.19 | 17.43 | 18.13 | 18.04 | 18.90 | 30.21 |
| PG($\gamma = 3.00$) | 17.76 | 18.13 | 18.58 | 18.69 | 19.73 | 30.35 |
| SB($\kappa = 3.00$) | 14.26 | 17.02 | 17.66 | 17.46 | 18.73 | 29.45 |
| SB($\kappa = 2.50$) | 15.43 | 17.44 | 18.31 | 17.68 | 19.16 | 29.83 |
| SB($\kappa = 2.00$) | 17.56 | 17.93 | 18.98 | 18.50 | 19.60 | 30.66 |
| SB($\kappa = 1.50$) | 22.09 | 20.10 | 20.86 | 20.70 | 22.04 | 32.68 |
| SB($\kappa = 1.25$) | 26.49 | 20.85 | 21.84 | 21.68 | 22.37 | 34.25 |
| SB($\kappa = 1.10$) | 34.85 | 25.10 | 26.18 | 25.94 | 28.28 | 31.46 |
| S_{opt} | 65.60 | 34.64 | 34.67 | 35.37 | 36.13 | 43.86 |

Table 3.11: Impact of partitioning strategies on the processing performance of Boolean queries on WIKI

Query-processing performance figures obtained for different *partitioning strategies* on our three datasets are given in Table 3.11–3.13. To obtain them, temporal coalescing was used on the WIKI and UKGOV dataset, but not on the NYT-30 dataset for which its application is not sensible.

For time-point Boolean queries we observe that our PG and SB partitioning strategies achieve wallclock times that are in between those measured on the indexes constructed using the extreme partitioning strategies S_{op} and P_{opt} . Apart from that, they are in sync with the reported expected processing costs that reflect query-processing performance for time-point queries on the indexes in

| | UKGOV | | | | | |
|-----------------------|-------|---------|--------|--------|--------|----------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| P_{opt} | 27.34 | 304.60 | 313.17 | 384.67 | 590.75 | 4,057.85 |
| PG($\gamma = 1.10$) | 28.57 | 309.70 | 280.36 | 334.03 | 538.54 | 3,493.99 |
| PG($\gamma = 1.25$) | 30.00 | 310.67 | 299.64 | 339.30 | 543.30 | 3,572.26 |
| PG($\gamma = 1.50$) | 32.08 | 315.20 | 296.10 | 345.95 | 551.16 | 3,590.42 |
| PG($\gamma = 2.00$) | 35.91 | 330.49 | 299.65 | 353.81 | 560.38 | 3,592.98 |
| PG($\gamma = 2.50$) | 40.95 | 347.83 | 316.26 | 370.59 | 577.35 | 3,605.77 |
| PG($\gamma = 3.00$) | 45.25 | 372.61 | 335.52 | 389.02 | 594.36 | 3,617.74 |
| SB($\kappa = 3.00$) | 35.46 | 334.47 | 315.30 | 359.77 | 562.14 | 3,554.64 |
| SB($\kappa = 2.50$) | 37.52 | 332.29 | 305.16 | 361.38 | 567.80 | 3,570.67 |
| SB($\kappa = 2.00$) | 40.67 | 345.35 | 314.12 | 370.34 | 575.87 | 3,577.23 |
| SB($\kappa = 1.50$) | 46.40 | 349.53 | 322.87 | 375.37 | 576.64 | 3,578.81 |
| SB($\kappa = 1.25$) | 54.50 | 382.36 | 348.70 | 397.66 | 604.86 | 3,591.55 |
| SB($\kappa = 1.10$) | 64.63 | 395.54 | 357.19 | 412.14 | 612.62 | 3,596.62 |
| S_{opt} | 74.58 | 406.75 | 414.22 | 478.69 | 675.11 | 4,141.95 |

Table 3.12: Impact of partitioning strategies on the processing performance of Boolean queries on UKGOV

| | NYT-30 | | | | | |
|-----------------------|--------|---------|--------|--------|--------|--------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| P_{opt} | 2.68 | 1.70 | 2.14 | 4.30 | 9.08 | 90.07 |
| PG($\gamma = 1.10$) | 2.79 | 1.65 | 1.95 | 3.10 | 5.12 | 45.08 |
| PG($\gamma = 1.25$) | 2.97 | 1.66 | 1.88 | 2.69 | 3.98 | 32.38 |
| PG($\gamma = 1.50$) | 3.29 | 1.74 | 1.90 | 2.62 | 3.58 | 26.58 |
| PG($\gamma = 2.00$) | 3.93 | 1.87 | 1.98 | 2.60 | 3.35 | 22.85 |
| PG($\gamma = 2.50$) | 4.52 | 1.97 | 2.09 | 2.62 | 3.35 | 21.90 |
| PG($\gamma = 3.00$) | 5.11 | 2.14 | 2.22 | 2.73 | 3.49 | 21.65 |
| SB($\kappa = 3.00$) | 4.08 | 1.95 | 2.09 | 2.79 | 3.70 | 25.56 |
| SB($\kappa = 2.50$) | 4.59 | 2.03 | 2.17 | 2.76 | 3.63 | 24.00 |
| SB($\kappa = 2.00$) | 5.61 | 2.22 | 2.31 | 2.92 | 3.61 | 22.37 |
| SB($\kappa = 1.50$) | 8.62 | 2.85 | 2.91 | 3.41 | 4.12 | 21.78 |
| SB($\kappa = 1.25$) | 14.45 | 3.97 | 4.20 | 4.80 | 5.25 | 22.00 |
| SB($\kappa = 1.10$) | 30.66 | 7.53 | 7.51 | 8.30 | 8.77 | 24.47 |
| S_{opt} | 397.52 | 82.88 | 84.28 | 85.11 | 85.39 | 96.41 |

Table 3.13: Impact of partitioning strategies on the processing performance of Boolean queries on NYT-30

general. When looking at larger query time-intervals, we observe that the differences in query-processing performance between the space-optimal partitioning strategy S_{opt} and its performance-optimal counterpart P_{opt} become smaller. On the WIKI dataset for queries with a year-granularity query time-interval S_{opt} even outperforms P_{opt} , as can be seen from Table 3.11. Thus, although the minimal number of postings is read from the index built using the performance-optimal partitioning strategy, as explained in Section 3.7, the cost of merging posting lists becomes an impeding factor for P_{opt} .

In contrast to that, our partitioning strategies PG and SB have to merge fewer posting lists than the performance-optimal P_{opt} and, at the same time, read fewer postings than the space-optimal S_{opt} . As a consequence, for larger query time-interval granularities, the indexes built using PG and SB often outperform the indexes built using the extreme partitioning strategies.

The impact that a combination of temporal coalescing and our partitioning strategies has on query-processing performance becomes clear, when comparing Table 3.10 against Table 3.11–Table 3.13. Thus, as one concrete example, on the WIKI dataset it takes 483 ms to process a time-point Boolean query on an index that is built without employing temporal coalescing and using the space-optimal partitioning strategy. When employing temporal coalescing in combination with our PG partitioning strategy for the parameter choice $\gamma = 1.50$, the same figure drops drastically to less than 16 ms.

Keyword Queries

Keyword queries are evaluated on the indexes with scalar payloads whose sizes are reported in Table 3.6 and Table 3.7.

We give query-processing performance figures obtained on indexes built for the WIKI and UKGOV dataset using the space-optimal partitioning strategy in combination with different choices of the *temporal coalescing* parameter ϵ alternatively without employing temporal coalescing in Table 3.14.

The figures shown support the following observations. For the parameter choice $\epsilon = 0.00$, wallclock times on the WIKI dataset increase when compared to the non-coalesced index (i.e., query-processing performance decreases). On the UKGOV dataset, in contrast, there is an improvement in query-processing performance for the same parameter choice. As argued above, due to the datasets' different characteristics, temporal coalescing has a lower impact on index size

| | WIKI | | | | | |
|----------------------|----------|---------|--------|--------|--------|--------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| Non-Coalesced | 1,445.80 | 485.71 | 483.77 | 500.97 | 510.92 | 740.43 |
| $\epsilon = 0.00$ | 1,384.12 | 568.70 | 564.64 | 591.33 | 595.53 | 854.27 |
| $\epsilon = 0.01$ | 745.02 | 349.17 | 347.96 | 362.72 | 377.29 | 639.33 |
| $\epsilon = 0.05$ | 749.65 | 338.39 | 335.93 | 351.92 | 365.90 | 629.05 |
| $\epsilon = 0.10$ | 745.02 | 332.81 | 332.12 | 346.17 | 359.73 | 614.44 |
| $\epsilon = 0.25$ | 741.72 | 330.41 | 329.96 | 342.99 | 356.49 | 613.93 |
| $\epsilon = 0.50$ | 741.12 | 330.92 | 329.12 | 341.47 | 355.84 | 608.26 |

| | UKGOV | | | | | |
|----------------------|----------|----------|----------|----------|----------|----------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| Non-Coalesced | 1,017.88 | 1,911.54 | 1,974.07 | 2,010.75 | 2,202.81 | 4,654.47 |
| $\epsilon = 0.00$ | 622.87 | 1,484.66 | 1,497.72 | 1,534.21 | 1,725.09 | 4,236.77 |
| $\epsilon = 0.01$ | 540.92 | 1,310.93 | 1,369.65 | 1,413.15 | 1,539.54 | 4,150.40 |
| $\epsilon = 0.05$ | 537.49 | 1,302.52 | 1,361.44 | 1,396.35 | 1,581.82 | 4,118.70 |
| $\epsilon = 0.10$ | 535.07 | 1,290.31 | 1,311.64 | 1,341.24 | 1,548.85 | 4,068.92 |
| $\epsilon = 0.25$ | 528.49 | 1,279.35 | 1,351.64 | 1,376.98 | 1,569.32 | 4,103.55 |
| $\epsilon = 0.50$ | 527.42 | 1,279.82 | 1,301.12 | 1,331.93 | 1,527.55 | 4,076.74 |

Table 3.14: Impact of temporal coalescing on the processing performance of keyword queries on WIKI and UKGOV

for the parameter choice $\epsilon = 0.00$ on the WIKI than on the UKGOV dataset. When processing keyword queries on Wikipedia using the index obtained for $\epsilon = 0.00$, we read almost as many postings as would be read from the non-coalesced index. In addition, as a penalty for query-processing performance, for each of them we consult the document-version dictionary when trying to de-coalesce the posting. For parameter choices $\epsilon > 0.00$, we see that temporal coalescing for scalar payloads improves query-processing performance consistently across datasets and query time-interval granularities. Again, as we discussed for Boolean queries above, for larger query time-interval granularities, the observed improvement is more substantial on WIKI than on UKGOV, due to the different time spans covered and the entailed effect on the selectivity of queries.

Query-processing performance figures obtained when using different partitioning strategies are given in Table 3.15–3.17. For the WIKI and UKGOV dataset temporal coalescing was employed using the parameter choice $\epsilon = 0.10$; for NYT-30 temporal coalescing was not employed for the reason mentioned above.

The figures shown reveal that for time-point keyword queries, our PG and SB partitioning strategies yield query-processing performance figures that behave as expected when we change γ and κ , respectively, and are close to the query-processing performance figures obtained for the performance-optimal P_{opt} . Further, they exhibit behavior in accordance with the query-independent expected processing costs that are meant to inversely query-processing performance of time-point keyword queries. For queries with larger query time-interval granularity, our observations are similar to the ones made above for Boolean queries, which is expected given that the partitioning strategies per se are not aware of the kind of posting payload. Thus, for larger query time-interval granularities, the differences in query-processing performance between the space-optimal S_{opt} and performance-optimal P_{opt} become smaller. For query time-intervals corresponding to days and weeks, our partitioning strategies achieve query-processing performance superior or comparable to the performance-optimal P_{opt} . For queries having a query time-interval at the month or year granularity, it is always a configuration of our PG or SB partitioning strategy that achieves the best query-processing performance.

| | WIKI | | | | | |
|-----------------------|--------|---------|--------|--------|--------|--------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| P_{opt} | 10.18 | 21.56 | 21.90 | 31.65 | 63.24 | 461.54 |
| PG($\gamma = 1.10$) | 11.49 | 21.78 | 22.49 | 30.53 | 58.23 | 398.05 |
| PG($\gamma = 1.25$) | 12.70 | 22.50 | 23.24 | 30.91 | 57.71 | 376.68 |
| PG($\gamma = 1.50$) | 14.75 | 23.27 | 24.08 | 32.00 | 57.05 | 372.37 |
| PG($\gamma = 2.00$) | 18.69 | 25.35 | 26.23 | 34.51 | 60.08 | 373.71 |
| PG($\gamma = 2.50$) | 22.41 | 27.43 | 28.16 | 36.95 | 61.10 | 381.24 |
| PG($\gamma = 3.00$) | 26.06 | 29.37 | 30.30 | 38.24 | 62.74 | 384.06 |
| SB($\kappa = 3.00$) | 17.54 | 24.52 | 25.54 | 33.96 | 59.44 | 378.12 |
| SB($\kappa = 2.50$) | 20.04 | 27.48 | 27.34 | 35.34 | 61.01 | 377.41 |
| SB($\kappa = 2.00$) | 25.03 | 28.61 | 28.91 | 37.19 | 62.02 | 381.28 |
| SB($\kappa = 1.50$) | 38.81 | 38.23 | 37.69 | 46.42 | 68.80 | 387.77 |
| SB($\kappa = 1.25$) | 63.53 | 48.54 | 50.87 | 58.15 | 80.68 | 404.63 |
| SB($\kappa = 1.10$) | 118.82 | 76.44 | 76.19 | 83.07 | 104.18 | 421.47 |
| S_{opt} | 745.02 | 332.81 | 332.12 | 346.17 | 359.73 | 614.44 |

Table 3.15: Impact of partitioning strategies on the processing performance of keyword queries on WIKI

| | UKGOV | | | | | |
|-----------------------|--------|----------|----------|----------|----------|----------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| P_{opt} | 30.02 | 281.39 | 289.59 | 341.48 | 598.31 | 4,118.69 |
| PG($\gamma = 1.10$) | 32.07 | 284.58 | 301.93 | 343.89 | 573.62 | 3,735.71 |
| PG($\gamma = 1.25$) | 35.29 | 291.28 | 310.16 | 355.70 | 581.18 | 3,665.00 |
| PG($\gamma = 1.50$) | 41.30 | 303.06 | 322.51 | 366.30 | 590.22 | 3,657.38 |
| PG($\gamma = 2.00$) | 53.03 | 333.82 | 348.36 | 391.87 | 625.20 | 3,706.48 |
| PG($\gamma = 2.50$) | 64.04 | 363.45 | 377.57 | 417.14 | 652.17 | 3,713.50 |
| PG($\gamma = 3.00$) | 74.94 | 388.79 | 402.92 | 442.21 | 670.97 | 3,760.01 |
| SB($\kappa = 3.00$) | 46.82 | 323.92 | 339.94 | 379.76 | 613.89 | 3,688.40 |
| SB($\kappa = 2.50$) | 52.71 | 350.34 | 369.64 | 406.42 | 633.97 | 3,702.49 |
| SB($\kappa = 2.00$) | 64.02 | 376.13 | 395.82 | 435.86 | 662.14 | 3,741.59 |
| SB($\kappa = 1.50$) | 93.37 | 457.55 | 469.58 | 510.26 | 739.67 | 3,801.08 |
| SB($\kappa = 1.25$) | 142.24 | 557.73 | 570.06 | 619.04 | 838.62 | 3,894.64 |
| SB($\kappa = 1.10$) | 234.19 | 779.70 | 791.49 | 829.00 | 1,047.90 | 4,046.13 |
| S_{opt} | 535.07 | 1,290.31 | 1,311.64 | 1,341.24 | 1,548.85 | 4,068.92 |

Table 3.16: Impact of partitioning strategies on the processing performance of keyword queries on UKGOV

| | NYT-30 | | | | | |
|-----------------------|--------|---------|--------|--------|--------|--------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| P_{opt} | 2.68 | 1.50 | 2.00 | 3.23 | 9.24 | 87.60 |
| PG($\gamma = 1.10$) | 2.79 | 1.59 | 1.84 | 2.38 | 5.16 | 44.96 |
| PG($\gamma = 1.25$) | 2.97 | 1.69 | 1.87 | 2.16 | 4.20 | 32.02 |
| PG($\gamma = 1.50$) | 3.29 | 1.79 | 1.93 | 2.21 | 3.71 | 25.86 |
| PG($\gamma = 2.00$) | 3.93 | 1.92 | 2.08 | 2.32 | 3.45 | 22.38 |
| PG($\gamma = 2.50$) | 4.52 | 2.04 | 2.29 | 2.42 | 3.62 | 21.12 |
| PG($\gamma = 3.00$) | 5.11 | 2.21 | 2.44 | 2.54 | 3.66 | 20.63 |
| SB($\kappa = 3.00$) | 4.08 | 2.00 | 2.19 | 2.37 | 3.76 | 24.53 |
| SB($\kappa = 2.50$) | 4.59 | 2.09 | 2.31 | 2.52 | 3.78 | 23.00 |
| SB($\kappa = 2.00$) | 5.60 | 2.43 | 2.57 | 2.74 | 3.96 | 21.64 |
| SB($\kappa = 1.50$) | 8.63 | 3.13 | 3.30 | 3.42 | 4.62 | 20.57 |
| SB($\kappa = 1.25$) | 14.49 | 4.60 | 4.61 | 4.87 | 5.77 | 20.89 |
| SB($\kappa = 1.10$) | 30.93 | 8.75 | 9.10 | 9.18 | 9.57 | 24.21 |
| S_{opt} | 397.52 | 91.02 | 92.44 | 90.94 | 92.15 | 103.49 |

Table 3.17: Impact of partitioning strategies on the processing performance of keyword queries on NYT-30

As a concrete figures demonstrating the impact that the combination of temporal coalescing and our partitioning strategies has, consider time-point keyword queries on the UKGOV dataset. When processed on a non-coalesced index built using S_{opt} their processing takes 1,912 ms on average. On our index obtained built using $PG(\gamma = 1.50)$ and employing temporal coalescing ($\epsilon = 0.10$) their processing takes as little as 303 ms on average.

Phrase Queries

Phrase queries, as the final query type considered in our experiments, are evaluated on the indexes with positional payloads whose sizes are given in Table 3.8 and Table 3.9.

Query-processing performance figures obtained for indexes built using the space-optimal partitioning strategy S_{opt} and employing *temporal coalescing* with different choices of the parameter η alternatively not employing temporal coalescing are given in Table 3.18.

On both datasets for the parameter choice $\eta = 1.00$ we observe a slight degradation of wallclock times in comparison to the non-coalesced index. For this parameter choice the lower number of coalesced postings read does hence not amortize the extra effort needed to decoalesce them using the in-memory document-version dictionary. For parameter choices $\eta > 1.00$, in contrast, the figures reveal consistent improvements in query-processing performance across both datasets and all query time-interval granularities. For the parameter choice $\eta = 1.50$, as the one that we use to study the impact of partitioning strategies below, time-point phrase queries are processed on average in 679 ms and 2,526 ms instead of 951 ms and 4,480 ms on the WIKI and UKGOV dataset, respectively.

For different *partitioning strategies* query-processing performance figures are given in Table 3.19– 3.21. To obtain these, on the WIKI and UKGOV dataset we employ temporal coalescing setting $\eta = 1.50$; on the NYT-30 dataset temporal coalescing is not employed.

The figures shown support our observations made for Boolean queries and keyword queries above. That is, for time-point phrase queries, PG and SB provide mean wallclock times that are in between those obtained from P_{opt} and S_{opt} . For time-interval queries, the differences in query-processing performance between P_{opt} and S_{opt} become smaller as we increase the query time-interval granularity. For year-granularity phrase queries, finally, we see configurations of our partitioning strategies on all datasets that outperform P_{opt} .

| | WIKI | | | | | |
|----------------------|----------|----------|----------|----------|----------|----------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| Non-Coalesced | 1,445.80 | 951.44 | 951.19 | 943.13 | 995.07 | 986.46 |
| $\eta = 1.00$ | 418.11 | 1,227.19 | 1,217.17 | 1,211.64 | 1,276.08 | 1,307.24 |
| $\eta = 1.50$ | 102.58 | 678.84 | 679.59 | 679.69 | 715.24 | 773.10 |
| $\eta = 2.50$ | 56.74 | 638.53 | 641.98 | 642.20 | 675.93 | 744.91 |
| $\eta = 5.00$ | 48.05 | 653.34 | 659.29 | 657.02 | 690.76 | 762.95 |

| | UKGOV | | | | | |
|----------------------|--------|----------|----------|----------|----------|----------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| Non-Coalesced | 620.58 | 4,480.46 | 4,687.23 | 4,491.38 | 4,776.01 | 5,474.52 |
| $\eta = 1.00$ | 247.60 | 5,504.39 | 5,620.84 | 5,530.45 | 5,747.33 | 6,692.16 |
| $\eta = 1.50$ | 66.27 | 2,526.49 | 2,547.09 | 2,558.81 | 2,624.36 | 3,812.22 |
| $\eta = 2.50$ | 48.54 | 2,324.66 | 2,286.91 | 2,345.87 | 2,382.28 | 3,718.90 |
| $\eta = 5.00$ | 44.48 | 2,330.45 | 2,282.28 | 2,351.37 | 2,378.61 | 3,708.58 |

Table 3.18: Impact of temporal coalescing on the processing performance of phrase queries on WIKI and UKGOV

| | WIKI | | | | | |
|---------------------|--------|---------|--------|--------|--------|--------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| P_{opt} | 4.41 | 11.43 | 11.90 | 13.84 | 22.34 | 140.93 |
| $PG(\gamma = 1.10)$ | 5.03 | 12.37 | 12.85 | 14.20 | 20.98 | 113.72 |
| $PG(\gamma = 1.25)$ | 5.46 | 13.47 | 13.93 | 15.37 | 22.14 | 116.92 |
| $PG(\gamma = 1.50)$ | 6.20 | 15.36 | 15.82 | 16.97 | 24.24 | 126.65 |
| $PG(\gamma = 2.00)$ | 7.55 | 19.18 | 19.51 | 20.34 | 26.91 | 130.98 |
| $PG(\gamma = 2.50)$ | 8.77 | 22.69 | 23.09 | 24.10 | 31.00 | 133.85 |
| $PG(\gamma = 3.00)$ | 9.95 | 25.42 | 25.97 | 26.75 | 32.43 | 136.02 |
| $SB(\kappa = 3.00)$ | 7.25 | 19.34 | 19.67 | 21.05 | 28.01 | 128.44 |
| $SB(\kappa = 2.50)$ | 8.13 | 20.78 | 21.04 | 22.13 | 28.45 | 131.05 |
| $SB(\kappa = 2.00)$ | 9.71 | 24.31 | 24.72 | 25.74 | 33.75 | 134.12 |
| $SB(\kappa = 1.50)$ | 14.03 | 32.48 | 32.79 | 34.63 | 38.97 | 141.76 |
| $SB(\kappa = 1.25)$ | 20.41 | 45.82 | 46.07 | 50.15 | 60.42 | 157.61 |
| $SB(\kappa = 1.10)$ | 31.76 | 69.50 | 69.69 | 71.04 | 81.20 | 173.04 |
| S_{opt} | 102.58 | 678.84 | 679.59 | 679.69 | 715.24 | 773.10 |

Table 3.19: Impact of partitioning strategies on the processing performance of phrase queries on WIKI

| | UKGOV | | | | | |
|---------------------|-------|----------|----------|----------|----------|----------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| P_{opt} | 16.96 | 528.50 | 516.92 | 548.67 | 670.77 | 2,930.19 |
| $PG(\gamma = 1.10)$ | 18.52 | 568.11 | 563.59 | 590.37 | 694.15 | 2,367.94 |
| $PG(\gamma = 1.25)$ | 20.25 | 620.38 | 616.08 | 649.01 | 762.57 | 2,535.54 |
| $PG(\gamma = 1.50)$ | 22.78 | 708.97 | 700.17 | 728.93 | 850.64 | 2,673.07 |
| $PG(\gamma = 2.00)$ | 27.33 | 861.82 | 854.87 | 885.16 | 1,012.00 | 2,805.04 |
| $PG(\gamma = 2.50)$ | 31.42 | 1,021.51 | 1,016.45 | 1,034.89 | 1,149.85 | 2,898.62 |
| $PG(\gamma = 3.00)$ | 34.70 | 1,144.48 | 1,138.94 | 1,155.99 | 1,260.85 | 3,028.47 |
| $SB(\kappa = 3.00)$ | 24.50 | 739.58 | 733.33 | 761.72 | 895.98 | 2,672.64 |
| $SB(\kappa = 2.50)$ | 26.16 | 797.56 | 793.84 | 840.46 | 946.43 | 2,777.00 |
| $SB(\kappa = 2.00)$ | 29.17 | 894.14 | 888.71 | 901.66 | 1,021.61 | 2,850.30 |
| $SB(\kappa = 1.50)$ | 35.73 | 1,172.85 | 1,169.16 | 1,192.55 | 1,313.89 | 3,038.06 |
| $SB(\kappa = 1.25)$ | 43.08 | 1,432.70 | 1,436.71 | 1,471.17 | 1,599.81 | 3,165.16 |
| $SB(\kappa = 1.10)$ | 53.24 | 2,126.01 | 2,121.27 | 2,144.86 | 2,215.62 | 3,648.32 |
| S_{opt} | 66.27 | 2,526.49 | 2,547.09 | 2,558.81 | 2,624.36 | 3,812.22 |

Table 3.20: Impact of partitioning strategies on the processing performance of phrase queries on UKGOV

| | NYT-30 | | | | | |
|---------------------|--------|---------|--------|--------|--------|--------|
| | EPC | MS (ms) | D (ms) | W (ms) | M (ms) | Y (ms) |
| P_{opt} | 2.68 | 2.16 | 3.05 | 4.40 | 13.96 | 123.16 |
| $PG(\gamma = 1.10)$ | 2.79 | 2.23 | 2.92 | 3.28 | 7.84 | 61.09 |
| $PG(\gamma = 1.25)$ | 2.97 | 2.40 | 3.02 | 3.07 | 6.18 | 42.99 |
| $PG(\gamma = 1.50)$ | 3.29 | 2.66 | 3.13 | 3.12 | 5.67 | 34.14 |
| $PG(\gamma = 2.00)$ | 3.93 | 3.06 | 3.67 | 3.51 | 5.63 | 29.39 |
| $PG(\gamma = 2.50)$ | 4.52 | 3.42 | 4.08 | 3.85 | 6.01 | 27.85 |
| $PG(\gamma = 3.00)$ | 5.11 | 3.80 | 4.52 | 4.15 | 6.16 | 27.34 |
| $SB(\kappa = 3.00)$ | 4.08 | 3.21 | 3.90 | 3.71 | 6.05 | 33.35 |
| $SB(\kappa = 2.50)$ | 4.59 | 3.54 | 4.20 | 3.97 | 6.33 | 31.16 |
| $SB(\kappa = 2.00)$ | 5.61 | 4.25 | 4.89 | 4.62 | 6.98 | 29.46 |
| $SB(\kappa = 1.50)$ | 8.60 | 6.18 | 6.93 | 6.48 | 8.01 | 28.39 |
| $SB(\kappa = 1.25)$ | 14.44 | 9.53 | 10.87 | 9.66 | 11.98 | 30.59 |
| $SB(\kappa = 1.10)$ | 30.64 | 22.33 | 22.94 | 22.81 | 22.49 | 41.20 |
| S_{opt} | 397.52 | 234.61 | 238.13 | 234.55 | 236.10 | 254.50 |

Table 3.21: Impact of partitioning strategies on the processing performance of phrase queries on NYT-30

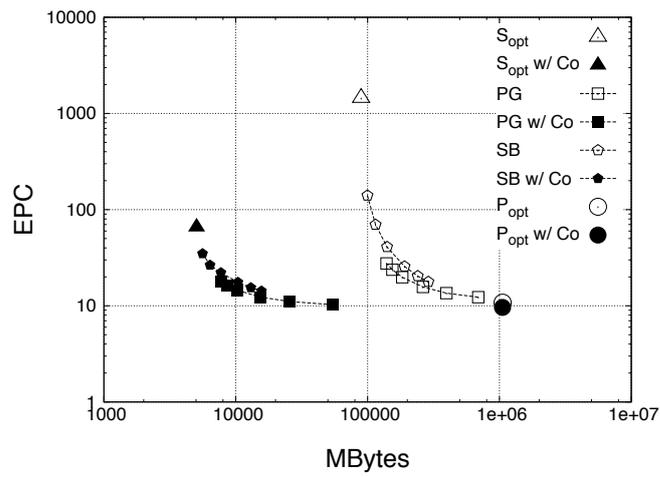
As a concrete figure demonstrating how effective a combination of temporal coalescing and our partitioning strategies is, consider time-point phrase queries on the UKGOV dataset. Processing them on a non-coalesced index built using S_{opt} takes 4,480 ms on average, as can be seen from Table 3.18. In contrast, on average they are processed in 894 ms on our index built using $SB(\kappa = 2.00)$ and employing temporal coalescing ($\eta = 1.50$), as Table 3.20 reveals.

Summary

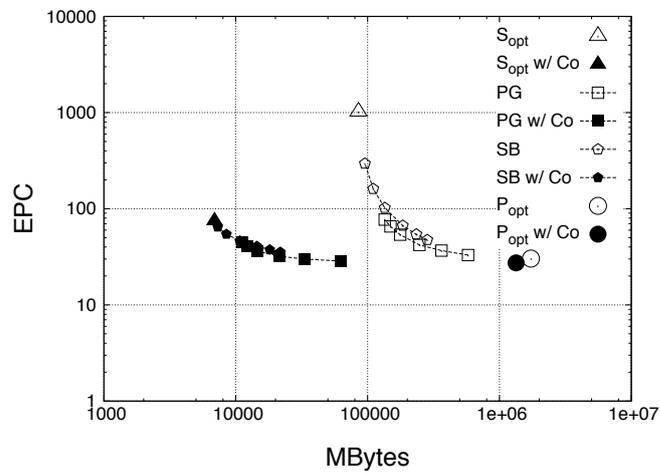
Having considered the aspects of index size and query-processing performance in isolation, to summarize our experimental evaluation, we now put the two together and examine their mutual trade-offs. Figure 3.15–3.17 plot index sizes (in MBytes) versus expected processing cost for Boolean queries, keyword queries, and phrase queries on our three datasets. For the WIKI and UKGOV dataset, for which employing temporal coalescing makes sense, the figures include plots both for non-coalesced and coalesced indexes. For scalar and positional payloads we once more employ the parameter choices $\epsilon = 0.10$ and $\eta = 1.50$ when employing temporal coalescing. We summarize our experimental evaluation with the following observations supported by the figures.

Our temporal partitioning techniques PG and SB allow to explore the spectrum between the extreme space-optimal and performance-optimal partitioning strategies. In doing, they make effective use of additional space, as can be seen from the convex shapes of their plots. To further illustrate this, consider that the rightmost point in all plots belonging to PG, which corresponds to $\gamma = 1.10$, consumes substantially less space than the performance-optimal partitioning strategy but achieves comparable performance. Similarly, the topmost point in the plots belonging to SB, which corresponds to $\kappa = 1.10$, consistently shows substantial improvements in expected processing cost over the space-optimal partitioning strategy while consuming only slightly more space.

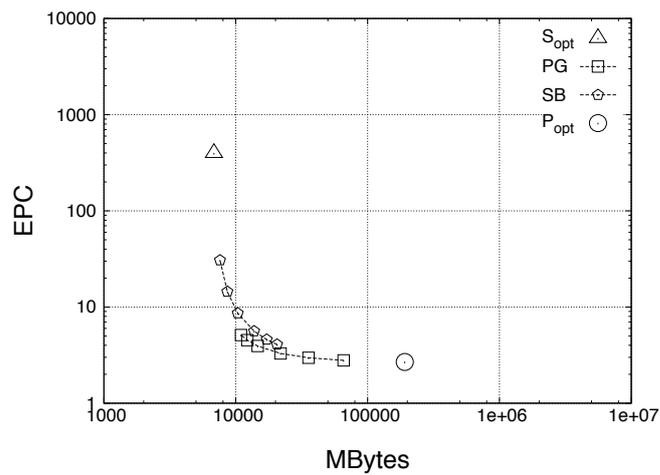
For all types of posting payloads temporal coalescing is highly effective and results in indexes that are both smaller in size and achieve lower expected processing cost than their non-coalesced counterparts. There is thus a synergetic effect between temporal coalescing and our partitioning strategies. For illustration, consider that all plots with solid markers in Figure 3.15–3.17, which correspond to coalesced indexes, are situated to the lower left of their non-coalesced analogs. Further, as experimentally examined above, temporal coalescing for



(a) WIKI

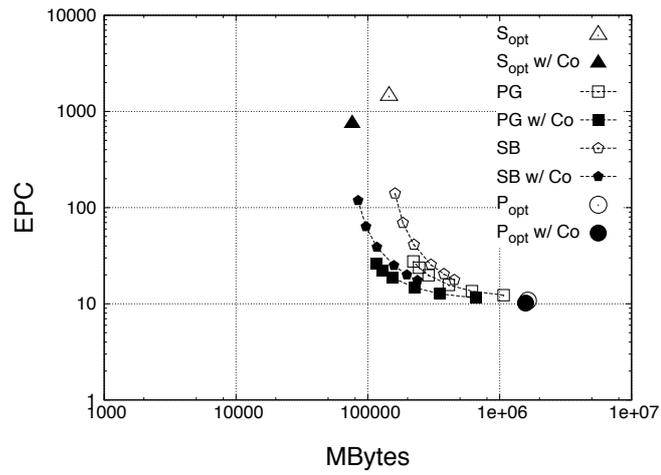


(b) UKGOV

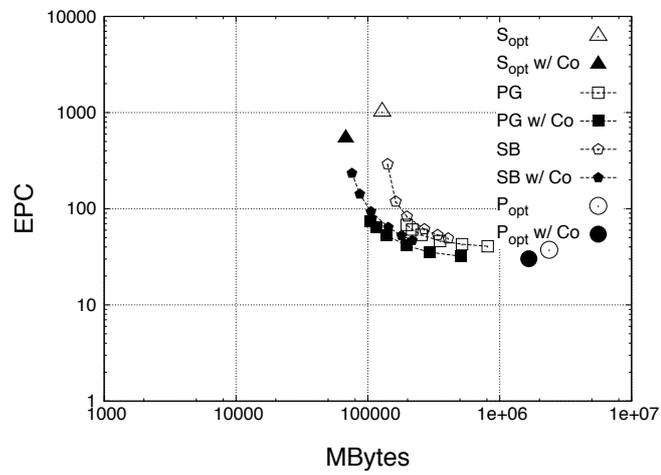


(c) NYT-30

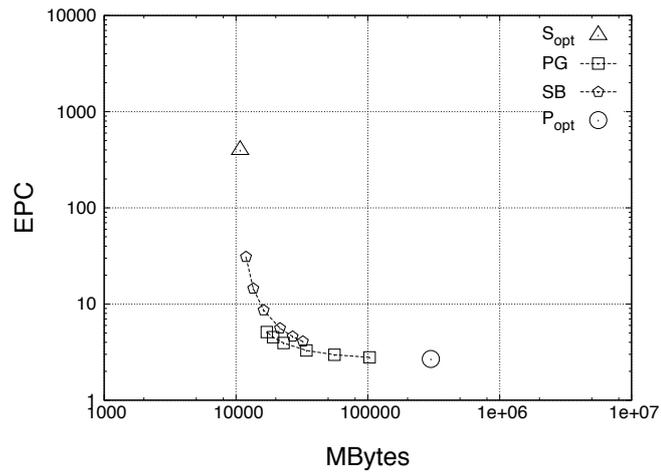
Figure 3.15: Index size and expected processing cost for Boolean queries on WIKI, UKGOV, and NYT-30



(a) WIKI

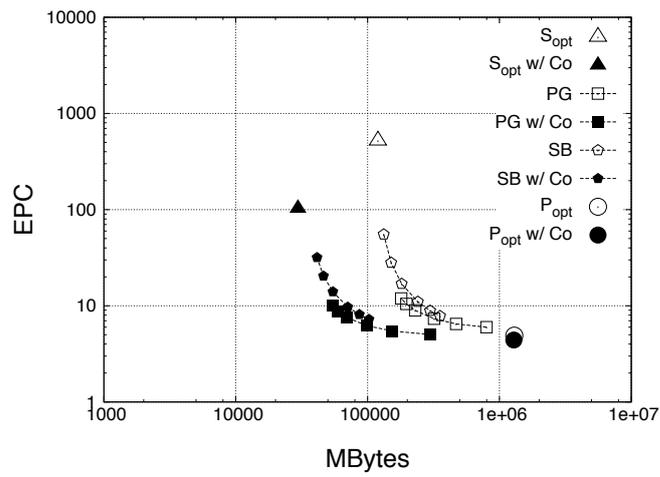


(b) UKGOV

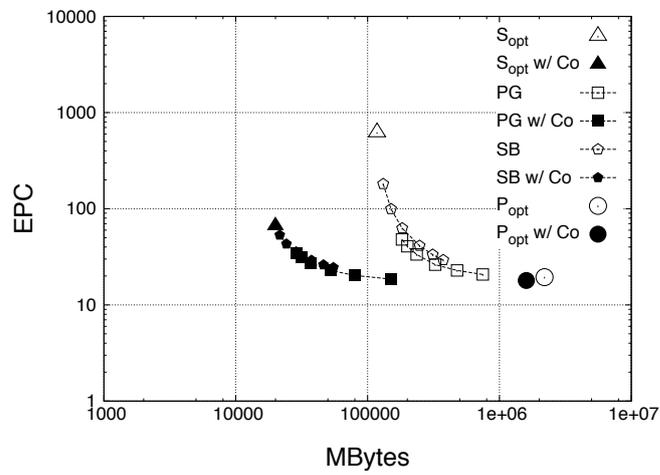


(c) NYT-30

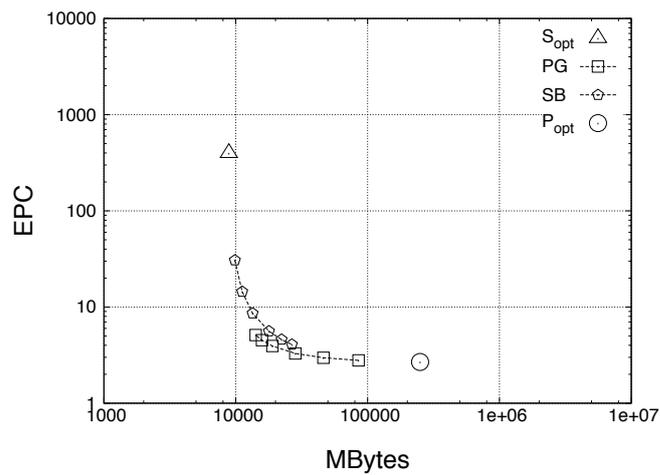
Figure 3.16: Index size and expected processing cost for keyword queries on WIKI, UKGOV, and NYT-30



(a) WIKI



(b) UKGOV



(c) NYT-30

Figure 3.17: Index size and expected processing cost for phrase queries on WIKI, UKGOV, and NYT-30

scalar payloads, achieves its considerable reduction in index size without a major distortion of query results.

Using a combination of temporal coalescing and our partitioning strategies is highly effective and achieves improvements both in terms of space consumption and query-processing performance over a naïve application of the inverted index. In detail, such a naïve application of the inverted index corresponds to not employing temporal coalescing and using the S_{opt} partitioning strategies, as we argued earlier in this chapter. From the figures belonging to the WIKI and UKGOV datasets it can be seen that we always produce a coalesced index (i.e., a point in the plots with solid markers) that is to the lower left of the baseline (i.e., the hollow triangle).

3.11 Discussion & Outlook

In this chapter, we have developed an efficient solution to time-travel text search in web archives and other versioned document collections. Comprehensive experiments on three real-world datasets showed that a combination of the proposed techniques can reduce index size by up to an order of magnitude while achieving nearly-optimal performance for time-point queries. In addition, we showed that time-interval queries can also be processed efficiently in practice using our techniques. For Boolean queries and phrase queries the proposed techniques produce accurate query results and our experiments demonstrated that the distortion induced to keyword query results is only minor.

Outlook

In our experiments different parameter choices for our partitioning strategies achieve the best performance for different query time-interval granularities. In practice, when arbitrary query time-interval granularities need to be supported, it may therefore be advisable to employ more than one parameter choice or partitioning strategy, thus introducing additional redundancy. Note that our query-processing techniques described in Section 3.5 are applicable to this scenario without any modification. Experimentally evaluating such a setup where more than one partitioning strategy is employed is an interesting open issue.

Beyond that, the present work opens up several interesting avenues for future research. Using the presented time-travel text search functionality to speed up or

improve text mining along the time axis (e.g., tracking sentiment changes in customer opinions) is one of them. A second direction that deserves more attention are retrieval models specifically designed for versioned document collections. In this work, we adapted an existing retrieval model, namely Okapi BM25, to deal with a versioned document collection. When processing a time-travel query, though, we may produce many nearly identical versions of the same document – an aggregation of versions belonging to the same document is desired. Ideally, the query result should hence contain for each document a set of its versions that trades off relevance and diversity, i.e., contains relevant document versions that are sufficiently different from each other. Third and finally, one may ask whether time-travel text search functionality as described in this work can also be implemented in a widely-distributed setting (e.g., a peer-to-peer system) – a problem that has recently attracted attention in [ABB⁺09a].

Chapter 4

Terminology Evolution in Web Archives

4.1 Motivation & Problem Statement

Web archives play a seminal role in preserving our cultural heritage for future generations. Among them, there are efforts such as the Internet Archive [IA], which has been archiving the publicly-accessible Web for more than a decade, but also other long-term document archives such as those operated by newspaper companies. These archives constantly grow in size as the Web evolves and new content is created, but also thanks to improved digitization techniques, which make it possible to add content that was originally published a long time ago. As a consequence, documents archived in these vast collections now cover at least decades, sometimes even centuries. For instance, the archives of The New York Times [NYTAA] range back until 1851; those of The Times [TIMES] even until 1785.

When searching these long-time archives, one challenging problem arises from the fact that terminology and general language use evolve constantly – a problem first identified in [TIR⁺08]. To illustrate this issue, consider the following two use cases:

- Carl Curious, a student of art, is writing a thesis about museums in Europe and searches a web archive for background information by issuing the keyword query `<saint petersburg museums>`. Not knowing that the city of Saint Petersburg was formerly known as Leningrad, Carl does not see those old but still highly relevant documents published in the 1970s with details on the Hermitage in Leningrad. These old documents would

typically not be retrieved by state-of-the-art retrieval methods that rely on a pure matching of keywords.

- Nelly Noise, a physician, is researching on hearing damage that can be caused by portable music players and issues the keyword query `<ipod hearing damage>`. Documents published in the 1980s that describe sudden deafness observed with heavy users of the Sony Walkman –the dominant portable music player at that time– would not be found.

As the two examples demonstrate, terminology evolution negatively affects retrieval effectiveness, and consequently user satisfaction, when searching web archives. This is because users typically employ current terminology when formulating queries – there is thus a widening gap between the terminology used in the queries and the terminology that was utilized in the past to write the now archived documents. Tackling this problem is essential in order to keep archived contents accessible and interpretable.

At a first glance, query expansion and refinement techniques [MRS08], as often employed in information retrieval to deal with what is called the word-mismatch problem between queries and documents, may seem like an adequate solution. These techniques modify the user's query typically by adding highly correlated terms. This is insufficient in our case for four reasons:

- There is *not necessarily a high correlation between* terms that were actively used in the past (e.g., `walkman`) and today's counterparts (e.g., `ipod`).
- Query reformulations should consider *entire phrases* even if the user types in only keywords (without phrase delimiters); for example `<middleware project costs>` could be reformulated into `<TP monitor man months>` for the 1980s, thereby mapping the phrase `TP monitor` to `middleware`.
- Query reformulations should be *sensitive to the order of query terms* – consider, as an example, the two queries `<caterpillar hearing damage>` and `<hearing caterpillar damage>` that, although containing the same set of terms, suggest very different meanings. Whereas for the former the user seems interested in hearing damage caused by caterpillars, for the latter the user is likely to be interested in court hearings related to a damaged caterpillar. Consequently, suitable reformulations for the two queries would ideally be very different.

- Independently substituting individual words by correlated words or phrases may lead to an undesired topic drift; for example, reformulating ⟨afro american president⟩ into ⟨african US chairman⟩ loses the user intention. Therefore, query reformulations should be *coherent* and thus consist of terms that can be sensibly put together.

For these four reasons, we take a different approach in this work. Given the user’s query, formulated using today’s terminology, our aim is to identify query reformulations that aptly capture the user’s information need employing terminology prevalent in the past. Such query formulations are insightful by themselves when presented to the user, who can then decide which of them should be issued to retrieve old documents that are highly relevant to the current information need. For the two use cases above, we would present Carl and Nelly with queries such as ⟨leningrad hermitage⟩ and ⟨walkman deafness⟩, respectively.

Finding adequate query reformulations also poses efficiency challenges for two reasons. First, the large scale of the web archives that we operate on, which comprise at least millions but often billions of documents. Second, users are impatient, so that achieving interactive response times at the order of at most a few seconds is mission critical.

Contributions

The work presented in this chapter makes the following contributions:

- A novel *measure of across-time semantic similarity* that assesses the degree of relatedness between two terms when used at different times.
- We develop a *query reformulation method* based on a Hidden Markov Model that considers the four aspects mentioned above and determines good query reformulations for a given user query.
- An efficient *implementation* of our approach is described that supports interactive response times.
- *Experiments* conducted on twenty years’ worth of New York Times articles demonstrating the usefulness of our approach.

Organization

The rest of this chapter is organized as follows. We put the work presented in context with prior work in Section 4.2. Our formal model and notation are introduced in Section 4.3. Section 4.4 delineates our measure of across-time semantic similarity. In Section 4.5, we describe our novel query reformulation method that avoids drifting towards incoherent queries. Section 4.6 details how the method can be implemented so as to achieve interactive response times. Our experiments described in Section 4.7 demonstrate the usefulness and efficiency of our approach. Finally, in Section 4.8, we conclude this chapter and point out future directions of research.

4.2 Related Work

We next put our work in context with prior research categorized as follows:

Query Expansion, Query Refinement, & Query Paraphrasing

Query expansion techniques [CTC05, TSW05, VRJ03, QF93, BSWZ03, XC96] address the so-called *word-mismatch problem* in information retrieval. To this end, the initial query is extended with terms that have been observed to co-occur often with the query terms (i) in the corpus as a whole (global techniques) or (ii) in a set of documents relevant to the initial query (local techniques). However, there are two key difference that distinguish existing query expansion and refinement techniques from our work. First, in their setting time is not explicitly taken into account, so that negative effects as the ones mentioned in the introduction can not be alleviated. Second, these techniques are typically implemented to be transparent to the user. Therefore, since the expanded query is not presented to the user, it is less critical if the query becomes unintuitive to the user or incoherent (e.g., by generating query expansions with dozens of keywords, but using disjunctive query semantics). Query paraphrasing techniques [BG09, ZRW02, ZR02] address this issue and seek to produce alternative queries that make sense to the user. Time, though, does not play a role in existing approaches. Further, the focus of query-paraphrasing techniques (e.g., Bernhard and Gurevych [BG09]) has often been on natural languages questions as issued to a question-answering system.

Cross-Language Information Retrieval

Prior work in cross-language information retrieval has addressed the question how a user query can be translated from one language to another [BS07, FB02, HCB⁺08, LJC05, MD05]. Closest to our work, the approach described by Federico and Bertoldi [FB02, BF04] employs a HMM for query translation. Although technically similar, there are two important differences that distinguish this line of research from our work. First, in cross-language information retrieval the existence of a dictionary that provides a few accurate translations for each of the original query terms is assumed – a task for which we have to resort to our measure of across-time semantic similarity. This also impacts efficiency, since we would consider more terms as across-time semantically similar than provided as a translation by a dictionary. Second, as discussed above for query-expansion techniques, their focus is not on producing queries that make sense to the user.

Information Retrieval on Historical Corpora

Information Retrieval on historical corpora has recently attracted some interest – a Dagstuhl seminar [BDFL07] dedicated to this problem was held in 2006. Existing approaches, such as Ernst-Gerlach and Fuhr [EGF06, EGF07] and Koolen et al. [KAKdR06], however, focus on changes in spelling over time. Although spelling changes are an interesting and important problem in their own right, the techniques proposed stay rather at the syntax level. They are therefore ill-suited to deal with cases such as our earlier *leningrad* example – where a syntactically completely different term like *saint petersburg* becomes part of prevalent language.

Suggesting Alternative Queries

Query suggestion is commonly used by current web search-engines to help users formulate their queries with less effort. In addition, these alternative query formulations are also very useful in the sponsored search domain - as a way to identify paid ad results that could be placed in the result rankings. Most of the state-of-the-art methods in this setting utilize the large-scale query log and/or click-through data to derive co-occurrence statistics and query reformulation behavior of users within a session [JRMG06, MZC08, AGMC09]. Unfortunately, when it comes to the problem addressed in this chapter, we not only struggle

with the paucity of query logs but also with the potential terminological variations within query logs.

4.3 Model

In this section, we formally define our time domain and collection model that will be used in the remainder of this chapter. Furthermore, we define the collection statistics that our techniques build upon.

4.3.1 Time Domain & Collection Model

We operate on a *timestamped document collection* \mathcal{D} . Each document $d^t \in \mathcal{D}$ bears a timestamp t that conveys its publication time. Timestamps are drawn from a *time domain* \mathcal{T} . We employ a discrete definition of time and assume the integers \mathbb{Z} as our time domain \mathcal{T} with timestamps $t \in \mathcal{T}$ denoting the number of time units (e.g., milliseconds or days) passed (to pass) since (until) a reference time-point (e.g., the UNIX epoch). The special value *now* always points to the current time and is assumed to be larger than any known timestamp. The techniques presented in the following are based on temporal partitions of the document collection defined by time intervals. Given a time interval T , we consider all documents d^t that were published during T , i.e., we demand $t \in T$.

4.3.2 Collection Statistics

We let \mathcal{V} denote the vocabulary of terms that occur in documents from our collection. Here, our notion of term includes keywords, but also multi-word expressions such as entity names. For a term $u \in \mathcal{V}$ we let $u@T$ denote the term when used during the time interval T . When writing $u@T$ and $u@R$, for instance, we thus refer to the same term u used during different time intervals. The number of occurrences of u in documents published during the time interval T is denoted as $\text{occ}(u@T)$. Further, given two terms u and v and a time interval T , the number of co-occurrences of the two terms in documents published during the time interval T is denoted as $\text{cooc}(u@T, v@T)$. In practice, co-occurrence counts are subject to further constraints. Thus, we may count a co-occurrence only (i) if u and v are within the same sentence, (ii) within a window of size ω , or (iii) we may take their order into account.

4.4 Across-Time Semantic Similarity

Having laid out our model and notation, we next introduce a method to assess the semantic similarity between two terms when used at different times, which will be a crucial building block for our query-reformulation technique.

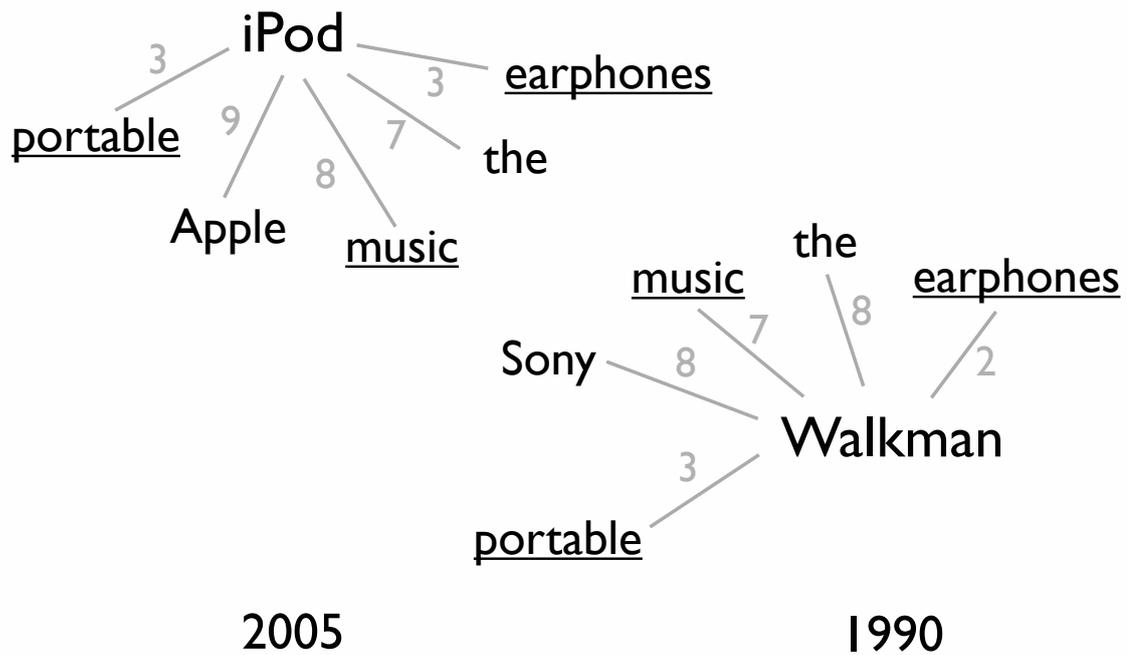


Figure 4.1: iPod@2005 and Walkman@1990 with frequently co-occurring terms

How can we assess the semantic similarity between two terms when used at different times? As a running example, consider the two terms iPod@2005 and Walkman@1990, for which we would like to assess a high degree of semantic similarity, since both devices were the dominant portable music players at the respective time. Figure 4.1 shows the two terms with their respective frequently co-occurring terms. As apparent from the figure, simple co-occurrence between the two terms, as often used by query expansion techniques, is not helpful here – neither of the terms occurs frequently together with the respective other term. However, notice the significant overlap between the terms that frequently co-occur with iPod@2005 and Walkman@1990 as, for instance, portable, music, and earphones. This significant overlap is a clear indication that the two terms are used in similar contexts at their respective time, which suggests the following:

Key Idea The degree of across-time semantic similarity between two terms $u@R$ and $v@T$ can be assessed by comparing the contexts –captured by co-occurrence statistics– in which u and v appear at time R and T , respectively.

This key idea is an adaptation of the *strong contextual hypothesis* discussed in Chapter 2 that has been around for decades [MC91, RG65].

In order to assess the degree of across-time semantic similarity, we propose a two-step generative model building on the above idea. In a first step, a term $w@T$ is randomly selected among terms that co-occur with the given $v@T$. Following that, in the second step, a term $u@R$ is selected among the terms that co-occur with the intermediate term w in documents published during R . In both steps, terms are selected with probability proportional to the observed co-occurrence statistics for the respective time. As an intuition behind the model, consider a user trying to find terms that are used in similar contexts as $v@T$ – a natural way to do so is to first identify terms that appear often together with $v@T$ (as *portable*, *music*, and *earphones* in our example), before examining their respective contexts to identify terms that appear often with all of them.

The probability of producing the term $u@R$ from the term $v@T$ according to the above generative model serves as our measure of across-time semantic similarity and is formally defined as follows:

Definition 4.1 (Across-time semantic similarity) Let $u@R$ and $v@T$ be two terms. We define the across-time semantic similarity between $u@R$ and $v@T$ as

$$P(u@R | v@T) = \sum_{w \in \mathcal{V}} P(u@R | w@R) \cdot P(w@T | v@T) \quad (4.1)$$

where $P(u@R | w@R)$ and $P(w@T | v@T)$ are estimated as

$$P(u@R | w@R) = \frac{\text{cooc}(w@R, u@R)}{\sum_{z \in \mathcal{V}} \text{cooc}(w@R, z@R)} \quad (4.2)$$

$$P(w@T | v@T) = \frac{\text{cooc}(v@T, w@T)}{\sum_{z \in \mathcal{V}} \text{cooc}(v@T, z@T)} \cdot \quad (4.3)$$

based on available co-occurrence statistics.

By our definition, across-time semantic similarity is not symmetric. Further, note that terms that co-occur frequently with $v@T$ but are frequent in general, such as the in Figure 4.1, have little impact on the assessed degree of similarity.

Since such terms that are generally frequent also co-occur with many terms, for them the value $P(u@R | w@R)$ is low, so that their effect on across-time semantic similarity is limited.

In practice, for a given $u@R$, one is often interested in efficiently determining the k terms $v@T$ having the highest degree of across-time semantic similarity. As we detail in Section 4.6 this can be accomplished efficiently using existing top- k query-processing techniques such as Fagin's NRA discussed in Chapter 2.

4.5 Query Reformulation

We now proceed to the core of this work and describe how queries can be reformulated to counter the negative effects induced by terminology evolution.

The problem addressed in this work can be formally stated as follows: We are given a user query $q = \langle q_1, \dots, q_m \rangle$ consisting of m query terms q_i , a reference time R , and a target time T . Our aim is to identify a query reformulation $q' = \langle q'_1, \dots, q'_m \rangle$ that aptly paraphrases the user's information need using the terminology valid at time T .

Notice that, in the rest of this section, for notational convenience, R and T are omitted for terms in the original query and query reformulations. Whenever we write q_i to refer to a term in the original query, the corresponding time is implicitly assumed to be the reference time R . Analogously, when referring to a term q'_i in a query reformulation, the corresponding time is assumed to be the target time T .

What makes a query reformulation q' one that aptly translates the user's information need? Consider again our example query $\langle \text{ saint_petersburg museum} \rangle @ 2005$, for which we would like to determine good query reformulations for the target time $T = 1990$. By means of this example, we next identify three desiderata for query reformulations.

Similarity

As mentioned earlier, a good query reformulations for our example query would be $\langle \text{ leningrad museum} \rangle @ 1990$. Notice that individual query terms in this reformulated query have a high degree of across-time semantic similarity with their counterparts in the original query. This would not be true for most other potential query reformulations such as $\langle \text{ economy europe} \rangle @ 1990$, for which the across-

time semantic similarity between terms and their counterparts in the original query will be much lower. Thus, a first desideratum for a reformulated query q' is that its query terms have high across-time semantic similarity with their respective counterparts in the original query q , i.e., we aim for high values of $P(q_i | q'_i)$.

Coherence

When taking into account only this first desideratum, though, we may end up with a nonsensical query reformulation. Consider $\langle \text{leningrad smithsonian} \rangle_{1990}$ as such a nonsensical reformulated query. This query meets our first desideratum, since the contained query terms leningrad_{1990} and $\text{smithsonian}_{1990}$ are semantically similar to their respective counterparts $\text{saint_petersburg}_{2005}$ and museum_{2005} in the original query. Putting the two terms leningrad and smithsonian together, though, makes little sense, given that the Smithsonian Institution, which comprises different museums, is located in Washington D.C. but not in Leningrad. As this example demonstrates, it is important to assert that putting the query terms q'_i next to each other makes sense, or to state it differently, whether the terms contained in the reformulated query are *coherent*. One way to do so is to examine whether the query terms co-occur frequently at the target time T , which can be done using the co-occurrence statistics that are at our disposal. Thus, as a second desideratum, we aim for high values of $P(q'_i | q'_{i-1})$ to assure a high level of coherence between adjacent terms in the reformulated query.

Popularity

Although similarity and coherence, as argued above, are crucial desiderata when determining good query reformulations, they still do not suffice. Consider the reformulated query $\langle \text{saarbruecken saarland_museum} \rangle_{1990}$ as an illustrating example. This query reformulation is reasonable with regard to similarity, since both Saarbruecken and Leningrad are cities and the Saarland Museum is a local museum. Also, with regard to coherence, the reformulated query is fine, given that the two terms saarbruecken and saarland_museum appear frequently together. It is unlikely, though, that this query reformulation is a satisfying reformulation that captures the user's information need, which could be to find about museums in large European cities. We should therefore take into account how

often query terms in the reformulated query occur at the target time, to avoid constructing whimsical query reformulations as the one above. To this end, we aim for terms q'_i in the reformulated query that occur frequently, thus having a high value $P(q'_i@T)$, which is defined as

$$P(u@T) = \frac{\text{occ}(u@T)}{\sum_{z \in \mathcal{V}} \text{occ}(z@T)} \quad (4.4)$$

for a term u and time T .

Now that we have identified the three desiderata *similarity*, *coherence*, and *popularity*, we next describe our approach to across-time query reformulation, which is based on a Hidden Markov Model (HMM) as defined in Chapter 2. By using a HMM we can take the order of query terms into account, which is crucial to produce good query reformulations, as we argued in the introduction.

The state space S of our HMM comprises all terms $v@T$; its alphabet Σ of output symbols contains all terms $v@R$. The initial state probability for the state $v@T$ (i.e., the probability to start in that state) is $P(v@T)$ as described above; it depends on the term's frequency of occurrence in documents published during T , and factors in the desideratum of popularity. The probability that the symbol $u@R$ is output from state $v@T$ is defined as $P(u@R | v@T)$, which is the across-time semantic similarity defined earlier, and thus covers the desideratum of similarity. The transition probability from $v@T$ to $w@T$ is defined as $P(w@T | v@T)$, which depends on how often the terms w and v co-occur in documents published during T , and factors in our desideratum of coherence.

Query reformulations $q' = \langle q'_1, \dots, q'_m \rangle$ thus correspond to state sequences in the above HMM. Good query reformulations according to our model can now be determined as those that have a corresponding state sequence with high probability of being traversed while outputting the original query $q = \langle q_1, \dots, q_m \rangle$. Formally this probability is given as

$$P(q | q') = P(q'_1) \cdot P(q_1 | q'_1) \cdot \prod_{i=2}^m P(q'_i | q'_{i-1}) \cdot P(q_i | q'_i). \quad (4.5)$$

In many cases, we would be interested not only in determining the best query reformulation, but in finding a set of k best query reformulations that the user can choose from. The best- k query reformulations can be identified efficiently using a combination of the Viterbi algorithm and A^* search, as described by Federico and Bertoldi [FB02, BF04] and proposed by Soong and Huang [SH91]. In the following, we give a concise description of this general-purpose technique

to identify the best- k state sequences in an HMM using our notation from Chapter 2. Algorithm 10 gives pseudo-code for the computation of the best- k state sequences.

In a first phase, the Viterbi algorithm is run. Using dynamic programming, as described in Chapter 2, the Viterbi algorithm initializes the maximum probability $\delta[i][j]$ of being in state s_i after j steps.

Following that, in a second phase, instead of performing the regular backtracking that determines the maximum probability state sequence, the algorithm performs an A^* search that leverages the information memoized by the Viterbi algorithm to determine the best- k state sequences.

A^* search, as described in Russel and Norvig [RN03], is a best-first search method. Adapted to our setting, the method traverses the space of state sequences that produce the observed output. A^* search builds up such state sequences in backward direction. In each iteration, new partial state sequences are generated by extending the currently most promising partial state sequence. For each partial state sequence $\langle s_{m-k}, \dots, s_m \rangle$ A^* search maintains two values, namely:

- $g(s_{m-k}, \dots, s_m)$ as the probability of traversing the partial state sequence while emitting the suffix $\langle \sigma_{m-k+1}, \dots, \sigma_m \rangle$ of the observed output.
- $h(s_{m-k}, \dots, s_m)$ as a heuristic estimate of the probability of getting to state s_{m-k} while emitting the prefix $\langle \sigma_1, \dots, \sigma_{m-k} \rangle$ of the observed output.

A heuristic for estimating $h(s_{m-k}, \dots, s_m)$ is called *admissible*, if it never underestimates the probability. If an admissible heuristic is used, A^* search is guaranteed to find state sequences in descending order of their probability. In our concrete setting, we leverage the information memoized by the Viterbi algorithm and set

$$h(s_{m-k}, \dots, s_m) = \delta(s_{m-k}, m - k). \quad (4.6)$$

Since $\delta(s_{m-k}, m - k)$ is the maximal probability of getting to state s_{m-k} after $(m - k)$ steps, it is guaranteed not to be an underestimate. Therefore, according to the above description, ours is an admissible heuristic. Note that

$$h(s_{m-k}, \dots, s_m) \cdot g(s_{m-k}, \dots, s_m) \quad (4.7)$$

is an upper bound on the probability of any state sequences that produces the observed output and has the suffix state sequence $\langle \sigma_{m-k}, \dots, \sigma_m \rangle$.

Algorithm 10: Computing the best-k state sequences

Data: HMM and observed output $\langle \sigma_1, \dots, \sigma_m \rangle$
Result: Best-k state sequences

```

1 /* Initialization */
2 partials = new MaxPriorityQueue()
3 bestK =  $\langle \rangle$ 
4 /* Phase 1: Run Viterbi algorithm and initialize  $\delta[i][j]$  */
5  $\delta[1..n][1..m]$ 
6 /* Phase 2: A* search */
7 for i = 1 to n do
8   if  $\delta[i][m] > 0$  then
9      $\rho = \langle s_i \rangle$ 
10    g = 1.0
11    h =  $\delta[i][m]$ 
12    t = m
13    partials.add( (g, h, t,  $\rho$ ), h · g )
14 while partials  $\neq \emptyset \wedge |\text{bestK}| < k$  do
15   (g, h, t,  $\langle q'_t, \dots, q'_m \rangle$ ) = partials.remove()
16   if t = 1 then
17     bestK = bestK  $\cup \{ \langle s_t, \dots, s_m \rangle \}$ 
18   else
19     for i = 1 to n do
20       if  $\delta[i][t-1] > 0$  then
21          $\rho' = \langle s_i \rangle \cup \langle s_t, \dots, s_m \rangle$ 
22          $g' = P(s_t | s_i) \cdot P(\sigma_t | s_t) \cdot g$ 
23          $h' = \delta[i][t-1]$ 
24          $t' = t - 1$ 
25         partials.add( (g', h', t',  $\rho'$ ), h' · g' )

```

The algorithm manages state sequences in a maximum priority queue `partials` based on their value $h(s_{m-k}, \dots, s_m) \cdot g(s_{m-k}, \dots, s_m)$. In each iteration, the most promising (i.e., highest probability) state sequence is removed from `partials`. If it is a complete state sequence of length m , it is added to the result list `bestK` that collects the best- k state sequences. Otherwise, if its length is less than m , new state sequences are derived from it that differ only in their initial state s_i and added to `partials`. Note that their priority $h' \cdot g'$ is less than or equal to the priority of the state sequence that they were generated from – this allows us to restrict the size of `partials` as k . The algorithm can thus safely terminate, once k complete state sequences have been added to `bestK`. This is because the last state sequence removed from `bestK` has probability larger than or equal to any state sequences that can be derived from state sequences still in `partials`.

The time complexity of the Viterbi algorithm is in $O(m \cdot n^2)$, its space complexity is in $O(m \cdot n)$. The space complexity of the A^* search is in $O(m \cdot k)$ – for keeping at most k state sequences of length less than or equal to m in `partials` and `bestK`, respectively. Its time complexity is in $O(n \cdot k^2 \cdot m)$ – the cost per iteration is in $O(n \cdot k)$ and it takes at most $O(k \cdot m)$ iterations to find the best- k state sequences. When making the reasonable assumption that $k \leq \sqrt{n}$, the time and space complexities of the overall method are dominated by those of the Viterbi algorithm and are thus in $O(m \cdot n^2)$ and $O(m \cdot n)$, respectively. This may seem prohibitive for our application, given that that $n = |V|$ is typically at the order of 10^7 and that we aim at interactive response times. Fortunately, though, for a given query q , large portions of the HMM can be disregarded during the computation, as we detail in the following section.

4.6 Implementation

So far, we have paid only little attention to how our methods can be implemented so as to achieve our objective of interactive response times.

Precomputations

In order to speed up the computation of across-time similarity scores and good query reformulations, we precompute values $P(u@T)$ and $P(u@T | v@T)$ for a fixed set of times T . In our concrete implementation times T correspond to calendar years, and we keep the precomputed values in main memory.

Pruning the State Space

As we explained in the previous section, the time and space complexity of the algorithm that we use to determine good query reformulations crucially depend on the number of states in our HMM. Fortunately, many of the states do not influence the result and can therefore be ignored during the computation. Thus, at query-processing time, we only have to consider a small part of the HMM that is sufficient to compute the accurate result. In detail, we can ignore all states corresponding to terms $v@T$ that fulfill

$$\forall q_i \in q : P(q_i@R | v@T) = 0 . \quad (4.8)$$

These states correspond to terms that can not output any of the original query terms and can therefore be safely ignored. This is because, by (4.5), state sequences that include such a state have zero probability of generating our original query.

Our implementation allows pruning the state space even further. In detail, for each of the original query terms $q_i@R$ we only consider the κ terms $v@T$ having highest probability of emitting the original query term – resulting in at most $m \cdot \kappa$ states in our HMM. By definition of our across-time semantic similarity measure, these κ terms can be identified efficiently using top-k query processing techniques such as the family of TA algorithms proposed by Fagin et al. [FLN03] (including the NRA algorithm that we described in Chapter 2). The reason for this is that across-time semantic similarity, according to Definition 4.1, is a monotonous aggregation function. This allows us to efficiently aggregate the precomputed probabilities $P(u@R | w@R)$ for all w such that $P(w@T | v@T) > 0$ and terminate early, once the κ best terms have been identified. To this end, the precomputed probabilities $P(u@R | w@R)$ are sorted in descending order and kept in main memory. However, in contrast to the pruning condition given above, this additional pruning is not safe and thus entails that only an approximate solution is produced.

Computing Query Reformulations

At query-processing time, good reformulations for a given query are then efficiently determined as follows. First, for each of the original query terms, we identify the κ (typically 1,000) terms $v@T$ to be included in the state space, as described above. Having built up the relevant portion of the state space, the Viterbi

algorithm is run. Our implementation looks up values $P(u@T | v@T)$ economically based on the rationale that we can avoid looking up values $P(u@T | v@T)$, if the state $v@T$ has zero probability of being visited. This is opposed to eagerly looking up values $P(u@T | v@T)$ for all term combinations. Finally, the best- k query reformulations are determined using A^* search as described above.

4.7 Experimental Evaluation

To evaluate the usefulness and efficiency of our approach, we conducted an experimental evaluation that is the subject of this section.

4.7.1 Setup & Dataset

Dataset

We employ the New York Times Annotated Corpus [NYT] as a dataset. This dataset contains a total of 1,855,656 newspaper articles published in New York Times between 1987 and 2007.

We further enriched the dataset, by annotating common phrases using the following “poor man’s” phrase extraction technique. For all term sequences consisting of up to eight terms and matching the title of an article in the English Wikipedia [WIKI], we add a special term to the document that represents the phrase. The rationale here is that by annotating common phrases, we get a hold on entity names, slogans, and other multi-word expressions. As an example, if a document contains the phrase “john lennon”, we add the special term `john_lennon` to the document, since there is a corresponding Wikipedia article about the musician John Lennon.

Collection statistics were precomputed for temporal partitions corresponding to calendar years. For the co-occurrence statistics we employ a value $\omega = 10$, take into account sentence boundaries, and disregard term order, i.e., whenever two terms u and v appear in the same sentence less than 10 terms apart, we count it as one co-occurrence. To remove noise and reduce the size of the data, values $\text{coc}(u@T, v@T)$ smaller than 5 are removed, i.e., we consider only pairs of terms u and v that occur at least five times together in documents published during the time interval T .

Setup

We implemented all methods in a small prototype system using Java 1.6 as a programming language. Data (including co-occurrence statistics, term frequencies etc.) was kept in an Oracle 10g relational database. The experiments described below were run on a single SUN V40z server-class machine having four AMD Opteron single-core CPUs, 16GB RAM, a large network-attached RAID-5 disk array, and running Microsoft Windows Server 2003.

Benchmark Terms & Queries

To evaluate our approach, we identified the two sets of benchmark terms and queries shown in Table 4.1 and Table 4.2, respectively. We choose such terms and queries that make sense only during a part of the time window that is covered by our document collection (e.g., nintendo ds), but for which we had a good idea about reasonable across-time semantically similar terms and good query reformulations, respectively.

| | Term |
|---|-----------------|
| 1 | pope_benedict |
| 2 | starbucks |
| 3 | mumbai |
| 4 | linux |
| 5 | mp3 |
| 6 | joschka_fischer |

Table 4.1: Benchmark terms used in our experimental evaluation

4.7.2 Across-Time Semantically Similar Terms

Across-time semantic similarity, as introduced above, plays a central role in our approach. Therefore, in this first part of our experimental evaluation, we examine how much sense the terms considered to have high across-time semantic similarity make. For each of our six benchmark terms Table 4.3 shows the ten terms considered most across-time semantically similar for the respectively specified reference and target time. From the results shown the following observations can be made:

| | Query |
|---|------------------------------------------------|
| 1 | <code><george_bush speech></code> |
| 2 | <code><colin_powell iraq></code> |
| 3 | <code><yahoo acquisition></code> |
| 4 | <code><airbus a380></code> |
| 5 | <code><christo gates></code> |
| 6 | <code><nintendo ds></code> |
| 7 | <code><tony_blair prime minister></code> |
| 8 | <code><angela_merkel berlin></code> |

Table 4.2: Benchmark queries used in our experimental evaluation

- For the term `pope_benedict` with reference time $R = 2005$ and target time $T = 1990$ our method identifies both terms as similar that (i) relate to Pope Benedict's former name Joseph Ratzinger but also (ii) to Pope John Paul II who was pope in 1990.
- Coffee-related terms and terms related to Dunkin Donuts (which was already popular in 1990) are brought up for the term `starbucks` with reference time $R = 2005$ and target time $T = 1990$.
- For the term `mumbai` with reference time $R = 2005$ and target time $T = 1990$ the city's name at the target time shows up among the highly-ranked terms.
- Different naming variations for the operating systems UNIX, DOS, and OS/2 that already existed at the target time $T = 1990$ are considered similar to the term `linux` with a reference time $R = 2005$ by our method.
- Terms relating to other music media such as `audio_cd` and `audio_tapes` are among the identified terms for the term `mp3` with reference time $R = 2005$ and target time $T = 1990$. However, also misleading terms such as `rockford_files`, which refers to a TV drama, are reported – because these terms are also often used in context with terms such as `files`.
- For the term `joschka_fischer` with reference time $R = 2005$ and target time $T = 1995$ our method brings up terms related to Klaus Kinkel, the German foreign minister in 1995, and other foreign ministers, which makes sense given that Joschka Fischer was foreign minister in 2005. Again, some of the

| | | |
|-----|----------------------------|------------------------------------|
| u | pope_benedict | starbucks |
| R/T | 2005 / 1990 | 2005 / 1990 |
| 1. | alexander_pope | dunkin_donuts |
| 2. | the_pope | dunkin |
| 3. | cardinal_ratzinger | donuts |
| 4. | joseph_cardinal_ratzinger | coffee_shops |
| 5. | pope_john_paul | cup_of_coffee |
| 6. | pope_john_paul_ii | a_cup_of_coffee |
| 7. | conservative_catholics | coffee_cup |
| 8. | polish-born | coffe_shop |
| 9. | irish_catholics | morning_coffee |
| 10. | frantisek_cardinal_tomasek | coffee_filter |
| u | mumbai | linux |
| R/T | 2005/1990 | 2005 / 1990 |
| 1. | air_india | unix_operating_system |
| 2. | bombay_india | unix_systems |
| 3. | krishnan | unix_international |
| 4. | india | the_operating_system |
| 5. | british_india | disk_operating_system |
| 6. | southern_india | dos_operating_system |
| 7. | north_india | operating_system |
| 8. | northern_india | operating_systems |
| 9. | passage_to_india | os |
| 10. | south_india | os_2 |
| u | mp3 | joschka_fischer |
| R/T | 2005 /1990 | 2005 / 1995 |
| 1. | audio_cd | klaus_kinkel |
| 2. | digital_audio | klaus |
| 3. | computer_files | bobby_fischer |
| 4. | s_files | stanley_fischer |
| 5. | the_rockford_files | searching_for_bobby_fischer |
| 6. | rockford_files | boris_spassky |
| 7. | audio_system | german_foreign_minister |
| 8. | audio_tapes | kinkel |
| 9. | audio_equipment | chinese_foreign_minister |
| 10. | audio_clips | foreign_affairs_minister_of_israel |

Table 4.3: Terms reported as most across-time semantically similar

terms are misleading and relate to chess player Bobby Fischer – because of a strong connection through the common last name and therefore frequent co-occurrence with fischer.

| | | |
|-----|--------------------------------------|-----------------------------------|
| q | ⟨ george_bush speech ⟩ | ⟨ colin_powell iraq ⟩ |
| R/T | 2005 / 1990 | 2005 / 1990 |
| 1. | ⟨george_bush speech⟩ | ⟨james_baker saddam_hussein⟩ |
| 2. | ⟨president_ronald_reagan excerpts⟩ | ⟨james_baker hussein⟩ |
| 3. | ⟨barbara_bush commencement⟩ | ⟨james_baker iraq⟩ |
| q | ⟨ yahoo acquisition ⟩ | ⟨ airbus a380 ⟩ |
| R/T | 2005 / 1995 | 2005 / 2000 |
| 1. | ⟨telesis_sbc⟩ | ⟨airbus industries⟩ |
| 2. | ⟨time_warner merger⟩ | ⟨a3xx superjumbo⟩ |
| 3. | ⟨america_online merger⟩ | ⟨airbus superjumbo⟩ |
| q | ⟨ christo gates ⟩ | ⟨ nintendo ds ⟩ |
| R/T | 2005 / 1995 | 2005 / 1990 |
| 1. | ⟨jeanne-claude christo⟩ | ⟨game_boy nintendo⟩ |
| 2. | ⟨christo reichstag⟩ | ⟨video-game nintendo⟩ |
| 3. | ⟨christo the_reichstag⟩ | ⟨galoob nintendo⟩ |
| q | ⟨ tony_blair prime minister ⟩ | ⟨ angela_merkel berlin ⟩ |
| R/T | 2005 / 1990 | 2005 / 1995 |
| 1. | ⟨margaret_thatcher prime minister⟩ | ⟨kohl helmut⟩ |
| 2. | ⟨yitzshak_shamir prime minister⟩ | ⟨christian_democratic_union kohl⟩ |
| 3. | ⟨vacek minister prime⟩ | ⟨helmut kohl⟩ |

Table 4.4: Top-3 across-time query reformulation results

4.7.3 Query Reformulation Results

In this second part of our experimental evaluation, we examine the quality of query reformulations produced by our method. For each term q_i in the original query we consider the up to $\kappa = 1,000$ terms having the highest probability of emitting q_i . Table 4.4 shows the query reformulations produced by our method for eight different queries using the reference time $R = 2005$ for all of them but varying the target time T . When determining the best query reformulations

using our method, we filter out query reformulations that are redundant in the sense that one of the query terms is a substring of another query term. These are rare but occur occasionally as an artifact of our corpus enrichment by phrase extraction. Notice that this filtering does not affect the algorithm, but can be done efficiently during the A* search phase. Thus, complete state sequences that correspond to a redundant query reformulation are discarded and not put into the result list. Response times when computing the query reformulations presented were in the order of 3–7 seconds, when using the setup described above. The query reformulations shown support the following observations:

- For the query `<george_bush speech>` with target time $T = 1990$ the first query reformulation does not alter the query at, which makes sense, considering that at that time George H. W. Bush was in office. Given the fact that Barbara Bush –the first lady at the time– gave the commencement speech at Wellesley College in 1990, also the query reformulation relating to her is sensible.
- Our method identifies queries related to James Baker, the United States Secretary of State at the target time $T = 1990$, for the query `<colin_powell iraq>`. This makes sense given that Colin Powell held the same office at the reference time.
- For the query `<yahoo acquisition>` with target time $T = 2000$ our method produces query reformulations most of which pertain to mergers and acquisitions among technology companies that happened during this period.
- Query reformulations yielded for the query `<airbus a380>` with a target time $T = 2000$ include early names and monikers of the A380 airplane.
- For the query `<christo gates>` with target time $T = 1995$ our method produces two query reformulations that relate to the wrapped Reichstag in Berlin – a piece of art that Christo and Jeanne-Claude created in 1995.
- The top-ranked query reformulation for `<nintendo ds>` with target time $T = 1990$ relates to the Nintendo Game Boy – a popular handheld video game device released in 1990.
- For the query `<tony_blair prime minister>` with target time $T = 1990$ the query reformulation `<margaret_thatcher prime minister>`, considered best

by our method, relates to Margaret Thatcher, Great Britain's prime minister at that time. Also the second query reformulation produced relates to a prime ministers of that time.

- Query reformulations relating to Helmut Kohl, the German Chancellor in 1990, are produced by our method for the query `<angela.merkel berlin>`.

Summary

The presented anecdotal results show that our methods make an important step in the right direction by producing insightful query reformulations that can counter negative effects induced by terminology evolution. Admittedly, as can also be seen from the results presented, there is room for future refinement. For instance, for the query reformulations produced for the query `<colin.powell iraq>`, a more *diverse* set of query reformulations may have been preferable.

4.8 Discussion & Outlook

In this chapter, we have made a first step toward countering the negative effects that terminology evolution induces in web archive search. We have proposed a novel measure of across-time semantic similarity, which is useful beyond the application considered in this work. Apart from that, we have developed an efficient technique to reformulate user queries. Our experimental evaluation on the New York Times Annotated Corpus, as a large-scale real-world dataset, demonstrates the usefulness of the proposed techniques and their efficiency.

Outlook

There is ample room for future research. Our method always produces query reformulations having the same number of terms as the original query. Replacing query terms one by one may not always be appropriate – consider the query `<compaq company history>`@1990 as an example, which would ideally be reformulated into `<hewlett packard company history>`@2009. In our concrete implementation, we employed phrase extraction techniques as a workaround to this problem. Further, we assumed a fixed set of temporal partitions for which co-occurrence statistics have been precomputed. Relaxing this assumption as to

allow for temporal partitions specified in an ad-hoc manner poses significant efficiency challenges and is therefore an interesting direction for future research.

Chapter 5

Retrieval Models for Temporal Information Needs

5.1 Motivation & Problem Statement

Many information needs have a temporal dimension, as expressed by a temporal phrase contained in the user's query and are best satisfied by documents that refer to a particular time. Existing retrieval models, however, often do not provide satisfying results for such *temporal information needs*, as the following examples demonstrate:

- A sports journalist, interested in FIFA World Cup tournaments during the 1990s, issues the query `fifa world cup 1990s`. Documents such as the New York Times articles shown in Figure 5.1 would often not be found by existing retrieval models, despite their obvious relevance to the journalist's information need. Similarly, a document stating *France won the FIFA World Cup in 1998* or a document published in 1998 mentioning *FIFA World Cup final in July* would be missed. This is because existing retrieval models are not aware of the semantic connections between the temporal expressions "in 1998" and "in July" contained in the documents and the user's query temporal expression "1990s".
- A historian, doing research on Christianization, issues the query `13th century crusades`. Documents with details on specific crusades, for instance, the Fourth Crusade that begun in 1202 would often not be among the retrieved results, unless they explicitly mention the 13th Century. Again, the reason is that existing retrieval models lack the knowledge about the

semantic connections between temporal expressions like “from 1202 until 1204” and “in 1202” contained in documents and temporal expression “13th century” contained in the query.



Figure 5.1: Documents from The New York Times relevant to the query fifa world cup 1990s likely to be missed by existing retrieval models

Improving retrieval effectiveness for such temporal information needs is an important objective for several reasons. First, a significant percentage of queries has temporal information needs behind them – about 1.5% of web queries were found to contain an explicit temporal expression (as Nunes et al. [NRD08] report) and about 7% of web queries have an implicit temporal intent (as Metzler et al. [MJPZ09] report). Note that these numbers are based on general web queries – for *specific domains* (e.g., news or sports) or *expert users* (e.g., journalists or historians) we expect a larger fraction of queries to have a temporal information need behind them. Second, thanks to improved digitization techniques and preservation efforts, many document collections, including the Web, nowadays

contain documents that (i) were *published a long time ago* and (ii) *refer to different times*. Consider, as one such document collection, the archive of The New York Times that covers the years 1851–2009. Articles in this archive provide a contemporary but also retrospective account on events during that time period. When searching these document archives, the temporal dimension plays an important role.

Temporal expressions are frequent across many kinds of documents and can be extracted and resolved with relative ease. However, it is not immediately clear how they should be integrated into a retrieval model. The key problem here is that the actual meaning of many temporal expressions is uncertain, or more specifically, it is not clear which exact time interval they actually refer to. As an illustration, consider the temporal expression “in 1998”. Depending on the context, it may refer to a particular day in that year, as in the above example of the FIFA World Cup final; or to the year as a whole, as in the sentence *in 1998 Bill Clinton was President of the United States*.

Our approach, in contrast to earlier work [ABB09b, BY05, KC05], considers this uncertainty. It integrates temporal expressions seamlessly into a language modeling approach, thus making them first-class citizens of the retrieval model.

Contributions

In detail, we make the following contributions in this chapter:

- We develop a *novel retrieval model* that integrates temporal expressions, in a principled manner, into a language modeling approach.
- Our *comprehensive experimental evaluation* that evaluates the proposed approach using two real-world datasets, namely the New York Times Annotated Corpus [NYT] and a snapshot of the English Wikipedia [WIKI], based on queries and corresponding relevance assessments that were obtained through the crowdsourcing platform Amazon Mechanical Turk [AMT].

Organization

The rest of this chapter is organized as follows. Section 5.2 puts our work in context with existing related research. In Section 5.3, we introduce our model and notation. Section 5.4 describes how temporal expressions can be integrated into a language modeling approach. Our experimental evaluation and its results

are described in Section 5.5. Finally, in Section 5.6, we conclude and point out promising open directions for future research.

5.2 Related Work

We now put our work in context with existing related research. The importance of temporal information for information retrieval is highlighted by Alonso et al. [AGBY07], who also mention the problem addressed in this chapter as one not yet satisfactorily supported by existing approaches. For our discussion of other related research, we broadly categorize it into the following three categories:

Time-Aware Retrieval Models

Li and Croft [LC03] and Dakka et al. [DGI08] both propose language models that take into account publication times of documents, in order to favor, for instance, more recent documents. Kanahuba and Nørnvåg [KN08] and de Jong et al. [dJRH05] employ language models to date documents, i.e., determine their publication time. Del Corso et al. [CGR05] address the problem of ranking news articles, taking into account publication times but also their interlinkage. Jones and Diaz [JD07] focus on constructing query-specific temporal profiles based on the publication times of relevant documents. Thus, all of the approaches mentioned are based on the publication times of documents. None of the approaches, though, considers temporal expressions contained in the documents.

Baeza-Yates [BY05], to the best of our knowledge, is the earliest approach that considers temporal expressions contained in documents for retrieval purposes. It aims at searching information that refers to the future. The proposed retrieval model is focused on confidences associated with statements about the future, thus favoring relevant documents that are confident about their predictions regarding a future time of interest. Kalczynski et al. [KC05] study the human perception of temporal expressions and propose a retrieval model for business news archives that takes into account temporal expressions. Arikan et al. [ABB09b] integrate temporal expressions into a language modeling approach but ignore the aspect of uncertainty. Metzler et al. [MJPZ09], most recently, identify so-called implicitly temporal queries and propose a method to bias ranking functions in favor of documents matching the user's implicit temporal intent.

Extraction of Temporal Expressions

The extraction of temporal expressions is a well-studied problem as discussed in more detail in Chapter 2. We represent temporal expressions as quadruples to capture their inherent uncertainty – a formal representation that we adopt from Zhang et al. [ZSYW08]. Koen and Bender [KB00] describe the Time Frames system that extracts temporal expressions and uses them to augment the user experience when reading news articles, for instance, by displaying a temporal context of concurrent events.

Several prototypes are available that make use of temporal expressions when searching the Web. TimeSearch [TSH] and Google’s Timeline View [GTV] are two notable examples among them. Details about their internals, though, have not been published.

Crowdsourcing for IR Evaluation

Crowdsourcing platforms such as Amazon Mechanical Turk [AMT] are becoming a common tool for conducting experiments in information retrieval. Amazon Mechanical Turk, as the best-known platform, allows requesters to publish so-called Human Intelligence Tasks (HITs), i.e., tasks that are hard for a computer but relatively easy for a human (e.g., determining the correct orientation of a photo). Apart from that, requesters can restrict the workers allowed to take up their HITs, for instance, based on their geographical location or depending on whether the worker passes a qualification test. On successful completion of a HIT, workers are paid a small reward, typically less than \$0.10. For a discussion of benefits and guidelines on how to use crowdsourcing platforms for experiments in IR, we refer to Alonso et al. [ARS08].

5.3 Model

In this section, we introduce our formal model and the notation that will be used throughout the rest of this chapter.

5.3.1 Time Domain & Temporal Expression Model

In this work, we apply a discrete notion of time and assume the integers \mathbb{Z} as our *time domain* \mathcal{T} with timestamps $t \in \mathcal{T}$ denoting the number of time units

(e.g., milliseconds or days) passed (to pass) since (until) a reference time-point (e.g., the UNIX epoch). These time units will be referred to as *chronons* in the remainder. Our formal representation of temporal expressions is defined as:

Definition 5.1 (Temporal Expression) *A temporal expressions T is formally represented as a quadruple*

$$T = (tb_l, tb_u, te_l, te_u) \quad (5.1)$$

with $tb_l, tb_u, te_l, te_u \in \mathcal{T}$. The temporal expression T can refer to any time interval $[b, e]$ such that $b \in [tb_l, tb_u]$, $e \in [te_l, te_u]$ and $b \leq e$.

In our representation tb_l and tb_u are respectively a lower bound and upper bound for the begin boundary of the time interval – marking the time interval’s earliest and latest possible begin time. Analogously, te_l and te_u are respectively a lower bound and upper bound for the end boundary of the time interval – marking the time interval’s earliest and latest possible end time. Since the time interval is not necessarily known exactly, we hence capture lower and upper bounds for its boundaries. To give a concrete example, the temporal expression “in 1998” from the introduction is represented as

$$(1998/01/01, 1998/12/31, 1998/01/01, 1998/12/31).$$

This representation thus captures the uncertainty inherent to many temporal expressions – a temporal expression T can refer to any time interval $[b, e]$ having a begin point $b \in [tb_l, tb_u]$ and an end point $e \in [te_l, te_u]$ along with the constraint $b \leq e$. We consider these time intervals thus as our *elementary units of meaning* in this work. In the remainder, when we refer to the temporal expression T , we implicitly denote the set of time intervals that T can refer to. Note that for notational convenience we use the format YYYY/MM/DD to represent chronons – their actual values are integers as described above.

5.3.2 Collection & Query Model

Let D denote our document collection. A document $d \in D$ is composed of its *textual part* d_{text} and its *temporal part* d_{time} . The textual part d_{text} is a bag of textual terms drawn from a vocabulary \mathcal{V} . The temporal part d_{time} is a bag of temporal expressions.

Analogously, a query q also consists of a textual part q_{text} and a temporal part q_{time} . We distinguish two modes of how we derive such a query from the

user's input, which differ in how they treat temporal expressions extracted from the input. In the *inclusive* mode, the parts of the user's input that constitute a temporal expression are still included in the textual part of the query. In the *exclusive* mode, these are no longer included in the textual part. Thus, for the user input `boston july 4 2002`, as a concrete example, in the inclusive mode we obtain $q_{\text{text}} = \{\text{boston, july, 4, 2002}\}$, whereas we obtain $q_{\text{text}} = \{\text{boston}\}$ in the exclusive mode.

5.4 Language Models for Temporal Information Needs

With our formal model and notation established, we now turn our attention to how temporal expressions can be integrated into a language modeling approach, and how we can leverage them to improve retrieval effectiveness for temporal information needs.

We use a query-likelihood approach and hence rank documents according to their estimated probability of generating the query. We assume that the textual and temporal part of the query q are generated independently from the corresponding parts of the document d , as captured in the following definition:

Definition 5.2 (Independent generation of query parts)

$$P(q | d) = P(q_{\text{text}} | d_{\text{text}}) \times P(q_{\text{time}} | d_{\text{time}}). \quad (5.2)$$

The first factor $P(q_{\text{text}} | d_{\text{text}})$ can be implemented using an existing text-based query-likelihood approach, e.g., the Ponte and Croft model [PC98]. In our concrete implementation, which we describe in detail in Section 5.5, we employ a unigram language model with Jelinek-Mercer smoothing.

For the second factor in the above equation, we assume that query temporal expressions in q_{time} are generated independently from each other, i.e.,

$$P(q_{\text{time}} | d_{\text{time}}) = \prod_{Q \in q_{\text{time}}} P(Q | d_{\text{time}}). \quad (5.3)$$

We use a two-step generative model to generate temporal expressions from a document d . In the first step, a temporal expression T is drawn at uniform random from the temporal expressions contained in the document. In the second step, a temporal expression is generated from the temporal expression T

just drawn. Under this model, the probability of generating the query temporal expression Q from document d is defined as follows:

Definition 5.3 (Generation of temporal expression from document)

$$P(Q | d_{\text{time}}) = \frac{1}{|d_{\text{time}}|} \sum_{T \in d_{\text{time}}} P(Q | T). \quad (5.4)$$

In the rest of this section, we describe two ways of defining the probability $P(Q | T)$. Like other language modeling approaches, our model is prone to the zero-probability problem – if one of the query temporal expressions has zero probability of being generated from the document, the probability of generating the query from this document is zero. To mitigate this problem, we employ Jelinek-Mercer smoothing, and estimate the probability of generating the query temporal expression Q from document d as

$$P(Q | d_{\text{time}}) = (1 - \lambda) \cdot \frac{1}{|D_{\text{time}}|} \sum_{T \in D_{\text{time}}} P(Q | T) + \lambda \cdot \frac{1}{|d_{\text{time}}|} \sum_{T \in d_{\text{time}}} P(Q | T) \quad (5.5)$$

where $\lambda \in [0, 1]$ is a tunable mixture parameter, and D_{time} refers to the temporal part of the document collection treated as a single very-large document.

Before giving two possible definitions of $P(Q | T)$, we identify the following requirements that any definition of $P(Q | T)$ must satisfy. Figure 5.2 illustrates these requirements – in the figure temporal expressions are represented as two-dimensional regions that encompass compatible combinations of begin point b and end point e .

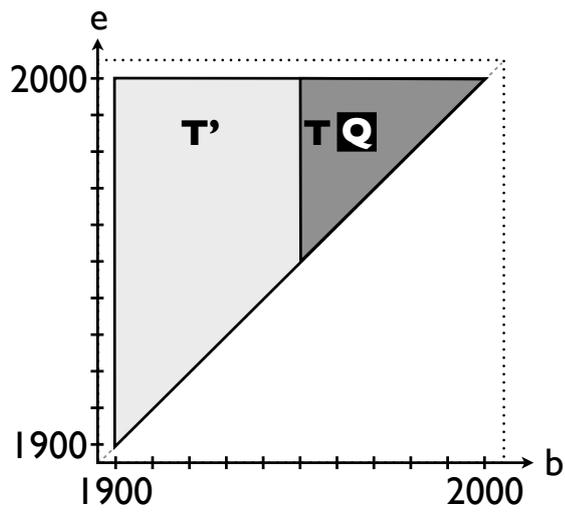
Definition 5.4 (Specificity) *Given two temporal expressions T and T' , we demand*

$$|T \cap Q| = |T' \cap Q| \wedge |T| \leq |T'| \Rightarrow P(Q | T) \geq P(Q | T'). \quad (5.6)$$

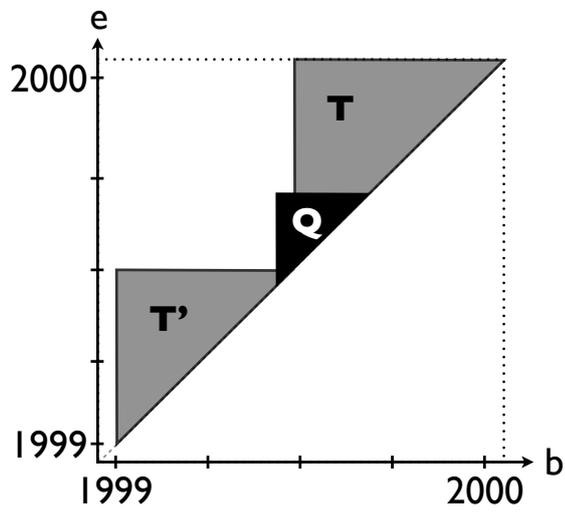
In other words, a query temporal expression is more likely to be generated from a temporal expression that closely matches it. Referring to Figure 5.2(a), the probability of generating Q (corresponding, for example, to “from the 1960s until the 1980s”) from T (corresponding, for example, to “in the second half of the 20th century”) is more than generating it from T' (corresponding, for example, to “in the 20th century”).

Definition 5.5 (Coverage) *Given two temporal expressions T and T' , we demand*

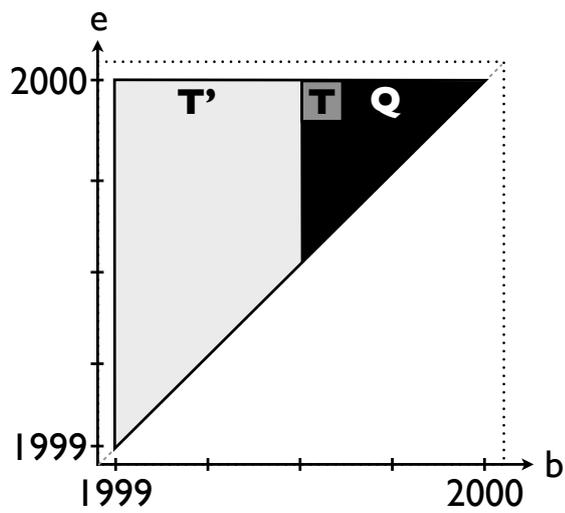
$$|T| = |T'| \wedge |T \cap Q| \leq |T' \cap Q| \Rightarrow P(Q | T) \leq P(Q | T'). \quad (5.7)$$



(a) Specificity



(b) Coverage



(c) Maximality

Figure 5.2: Three requirements for a generative model

In this requirement, we capture the intuition that a larger overlap with the query temporal expression is preferred. In Figure 5.2(b), the overlap of Q (corresponding, for instance, to “in the summer of 1999”) with T (corresponding, for instance, to “in the first half of 1999”) is more than the overlap with T' (corresponding, e.g., to “in the second half of 1999”). Therefore, the latter temporal expression is preferable and should have a higher probability of generating Q .

Definition 5.6 (Maximality) $P(Q | T)$ should be maximal for $T = Q$, i.e.,

$$T \neq Q \Rightarrow P(Q | T) \leq P(Q | Q) . \quad (5.8)$$

This requirement captures the intuition that the probability of generating a query temporal expression from a temporal expression matching it exactly must be the highest. As shown in Figure 5.2(c), the probability of generating Q (corresponding, for example, to “in the second half of 1999”) from itself should be higher than the probability of generating it from T (corresponding, for example, to “from July 1999 until December 1999”) or T' (corresponding, for example, to “in 1999”).

5.4.1 Uncertainty-Ignorant Language Model

Our first approach, further referred to as LMT, ignores the uncertainty inherent to temporal expressions. According to the following definition, a temporal expression T can only generate itself.

Definition 5.7 (LMT) Let Q and T be temporal expressions, LMT defines the probability of generating Q from T as

$$P(Q | T) = \mathbf{1}(T = Q) , \quad (5.9)$$

where $\mathbf{1}(T = Q)$ is an indicator function whose value assumes 1 iff $T = Q$ (i.e., $tb_l = qb_l \wedge tb_u = qb_u \wedge te_l = qe_l \wedge te_u = qe_u$).

The approach thus ignores uncertainty, since it misses the fact that a temporal expression T and a query temporal expression Q may refer to the same time interval, although $T \neq Q$. As we show next LMT meets our above requirements.

Theorem 5.1 LMT meets the requirements of specificity, coverage, and maximality.

Proof of Theorem 5.1 We prove that specificity holds by showing the inverse direction

$$\begin{aligned}
 P(Q | T) < P(Q | T') &\Leftrightarrow Q \neq T \wedge Q = T' \\
 &\Leftrightarrow (|T \cap Q| \neq |T' \cap Q| \wedge Q \neq T \wedge Q = T') \vee \\
 &\quad (|T \cap Q| = |T' \cap Q| \wedge Q \neq T \wedge Q = T') \\
 &\Rightarrow |T \cap Q| \neq |T' \cap Q| \vee |T| > |T'|.
 \end{aligned}$$

We prove that coverage holds by showing the inverse direction

$$\begin{aligned}
 P(Q | T) > P(Q | T') &\Leftrightarrow Q = T \wedge Q \neq T' \\
 &\Leftrightarrow (|T| \neq |T'| \wedge Q = T \wedge Q \neq T') \vee \\
 &\quad (|T| = |T'| \wedge Q = T \wedge Q \neq T') \\
 &\Rightarrow |T| \neq |T'| \vee |T \cap Q| > |T' \cap Q|.
 \end{aligned}$$

Finally, maximality holds for LMT, since

$$T \neq Q \Rightarrow P(Q | T) = 0 < P(Q | Q) = 1.$$

□

Despite its simplicity the approach still profits from the extraction of temporal expressions. To illustrate this, consider the two temporal expressions “in the 1980s” and “in the '80s”. Both share the same formal representation in our model, so that LMT can generate a query containing one of them from a document containing the other. A text-based approach that does not pay special attention to temporal expressions, in contrast, would not be aware of the semantic connection between the textual terms '80s and 1980s.

5.4.2 Uncertainty-Aware Language Model

As explained in the introduction, for many temporal expressions the exact time interval that they refer to is uncertain. Our second approach LMTU explicitly considers this uncertainty. In detail, we define the probability of generating Q from the document d as

$$P(Q | T) = \frac{1}{|Q|} \sum_{[q_b, q_e] \in Q} P([q_b, q_e] | T), \quad (5.10)$$

where the sum ranges over all time intervals included in Q . The approach thus assumes equal likelihood for each time interval $[q_b, q_e]$ that Q can refer to. Intuitively, each time interval that the user may have had in mind when uttering Q is assumed equally likely. Recall that $|Q|$ denotes the huge but finite total number of such time intervals.

The probability of generating the time interval $[q_b, q_e]$ from a temporal expression T is defined as

$$P([q_b, q_e] | T) = \frac{1}{|T|} \mathbb{1}([q_b, q_e] \in T), \quad (5.11)$$

where $\mathbb{1}([q_b, q_e] \in T)$ is an indicator function whose value is 1 iff $[q_b, q_e] \in T$. For T we thus also assume all time intervals that it can refer to as equally likely. Putting the two equations together we obtain

$$P(Q | T) = \frac{1}{|Q|} \sum_{[q_b, q_e] \in Q} \frac{1}{|T|} \mathbb{1}([q_b, q_e] \in T), \quad (5.12)$$

which can be simplified into the following compact definition of our uncertainty-aware language model.

Definition 5.8 (LMTU) *Let Q and T be temporal expressions, LMTU defines the probability of generating Q from T as*

$$P(Q | T) = \frac{|T \cap Q|}{|T| \cdot |Q|}. \quad (5.13)$$

Both Q and T are inherently uncertain. It is not clear which time interval the user issuing the query and author writing the document had in mind when uttering Q and T , respectively. Having no further information, our model assumes equal likelihood for all possible time intervals that Q and T can refer to, respectively.

Theorem 5.2 *LMTU meets the requirements of specificity, coverage, and maximality.*

Proof of Theorem 5.2 *For LMTU specificity and coverage follow immediately from Definition 5.8. To see that maximality holds, observe that $P(Q | Q) = 1/|Q|$ according to the above equation. Maximality then follows from the fact that*

$$T \neq Q \Rightarrow |T \cap Q|/|T| \leq 1.$$

□

Efficient Computation

For the practical applicability of this model, one important issue that needs to be addressed is the efficient computation of $P(Q | T)$ as defined above. Naïvely enumerating all time intervals that T and Q can refer to, before computing $|T \cap Q|$ is clearly not a practical solution. Consider again the temporal expression

$$(1998/01/01, 1998/12/31, 1998/01/01, 1998/12/31).$$

For a temporal resolution with chronons corresponding to days the total number of time intervals that this temporal expression can refer to is 66,795. When we make the granularity more fine-grained such that chronons correspond to hours, this number becomes 38,373,180. Fortunately, there is a more efficient way to compute $P(Q | T)$, as we explain next.

Theorem 5.3 *The probability $P(Q | T)$ according to Definition 5.8 can be computed efficiently without enumerating all time intervals that Q respectively T can refer to.*

Proof of of Theorem 5.3 *We first show that $|T|$ can be computed efficiently for any temporal expression T . Let $T = (tb_l, tb_u, te_l, te_u)$ be a temporal expression, we distinguish two cases:*

(i) *if $tb_u \leq te_l$ then $|T|$ can simply be computed as*

$$(tb_u - tb_l + 1) \cdot (te_u - te_l + 1),$$

since any begin point b is compatible with any end point e , otherwise,

(ii) *if $tb_u > te_l$ then $|T|$ can be computed as*

$$|T| = \sum_{tb=tb_l}^{tb_u} (te_u - \max(tb, te_l) + 1), \quad (5.14)$$

which captures that only end points $e \geq \max(b, te_l)$ are compatible with a fixed begin point b . Recall that we assume $tb_u > te_l$. This can be simplified into a

closed-form expression as follows:

$$\begin{aligned}
 |\mathbb{T}| &= \sum_{tb=tb_l}^{tb_u} (te_u - \max(tb, te_l) + 1) \\
 &= \sum_{tb=tb_l}^{te_l} (te_u - \max(tb, te_l) + 1) + \sum_{tb=te_l+1}^{tb_u} (te_u - \max(tb, te_l) + 1) \\
 &= (te_l - tb_l + 1) \cdot (te_u - te_l + 1) + \sum_{tb=te_l+1}^{tb_u} (te_u - tb + 1) \\
 &= (te_l - tb_l + 1) \cdot (te_u - te_l + 1) + \sum_{c=1}^{tb_u - te_l} (te_u - c - te_l + 1) \\
 &= (te_l - tb_l + 1) \cdot (te_u - te_l + 1) \\
 &\quad + (tb_u - te_l) \cdot (te_u - te_l + 1) - \sum_{c=1}^{tb_u - te_l} c \\
 &= (te_l - tb_l + 1) \cdot (te_u - te_l + 1) \tag{5.15} \\
 &\quad + (tb_u - te_l) \cdot (te_u - te_l + 1) - 0.5 \cdot (tb_u - te_l) \cdot (tb_u - te_l + 1) .
 \end{aligned}$$

Let $Q = (qb_l, qb_u, qe_l, qe_u)$ be a query temporal expression. We can compute $|Q|$ using our preceding arguments. For computing $|Q \cap \mathbb{T}|$ observe that each time interval $[b, e] \in Q \cap \mathbb{T}$ fulfills $b \in [tb_l, tb_u] \cap [qb_l, qb_u]$ and $e \in [te_l, te_u] \cap [qe_l, qe_u]$.

This completes our proof, since we can compute $|\mathbb{T} \cap Q|$ by considering the following temporal expression

$$(\max(tb_l, qb_l), \min(tb_u, qb_u), \max(te_l, qe_l), \min(te_u, qe_u)) .$$

□

Thus, we have shown that the generative model underlying LMTU allows for efficient computation. When processing a query with a query temporal expression Q , we need to examine all temporal expressions T with $T \cap Q \neq \emptyset$ and the documents that contain them. This can be implemented efficiently by keeping a small inverted index in main memory that keeps track of the documents that contain a specific temporal expression. Its lexicon, which consists of the known temporal expressions, can be organized using an interval tree to support the efficient identification of qualifying temporal expressions via interval intersection.

5.5 Experimental Evaluation

This section describes our experimental evaluation of the proposed approach.

5.5.1 Setup & Datasets

Methods under Comparison

We compare the following methods:

- $LM(\gamma)$ – Unigram language model with Jelinek-Mercer smoothing.
- $LMT-IN(\gamma, \lambda)$ – Uncertainty-ignorant method using inclusive mode.
- $LMT-EX(\gamma, \lambda)$ – Uncertainty-ignorant method using exclusive mode.
- $LMTU-IN(\gamma, \lambda)$ – Uncertainty-aware method using inclusive mode.
- $LMTU-EX(\gamma, \lambda)$ – Uncertainty-aware method using exclusive mode.

Apart from our baseline LM, we thus consider all four combinations of (a) inclusive vs. exclusive mode (i.e., whether query terms constituting a temporal expression are part of q_{text}) and (b) uncertainty-ignorant vs. uncertainty-aware definition of $P(Q|T)$. The mixture parameters γ and λ control the Jelinek-Mercer smoothing used when generating the textual part and the temporal part of the query, respectively. We consider values in $\{0.25, 0.5, 0.75\}$ for each of them, giving us a total of 39 method configurations under comparison. Further, note that our baseline LM, which is not aware of temporal expressions, always only considers q_{text} as determined using the inclusive mode, i.e., containing all terms from the user’s input.

Implementation Details

We implemented all methods in Java 1.6 keeping data in an Oracle 11g database. When indexing the two document collections, we did not remove stopwords nor apply lemmatization/stemming. Temporal expressions were extracted using TARSQI [VMS⁺05]. 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 [TIMEML] markup language. Building on TARSQI’s output, we extracted range temporal expressions such as “from

1999 until 2002”, which TARSQI does not yet support. Further, we added each article’s publication date as an additional temporal expression. We map temporal expressions to our quadruple representation using milliseconds as chronons and the UNIX epoch (i.e., midnight of January 1, 1970) as our reference timepoint. All experiments were run on a single SUN V40z server-class machine having four AMD Opteron single-core CPUs, 16GB RAM, a large network-attached RAID-5 disk array, and running Microsoft Windows Server 2003.

Datasets

We use the following two publicly-available datasets for our experimental evaluation:

- *The New York Times Annotated Corpus* [NYT] (NYT) that contains 1,855,656 articles published in New York Times between 1987 and 2007.
- *The English Wikipedia* [WIKI] (WIKI) as of July 7, 2009 that contains a total of 2,955,294 encyclopedia articles.

| | NYT | WIKI |
|--------------------------------------------------|-----------|-----------|
| # Documents | 1,855,656 | 2,955,294 |
| Document length in words (μ) | 691.79 | 617.18 |
| Document length in words (σ) | 722.88 | 1101.51 |
| # Temporal expressions per document (μ) | 6.35 | 12.91 |
| # Temporal expressions per document (σ) | 5.86 | 33.20 |

Table 5.1: Dataset statistics (with mean μ and standard deviation σ)

Table 5.1 shows additional statistics about the two datasets. From the figures we observe that the mean document length is similar for both datasets. Documents from WIKI, on average, contain more than twice as many temporal expressions as documents from NYT.

Queries

Since we target a specific class of information needs, query workloads used in benchmarks like TREC [TRE] are unemployable in our setting. Search-engine query logs, on the other hand, as a second valuable source of realistic queries, are typically not publicly available. To assemble a query workload that captures

Complete a Time-Related Query

We are interested in exploring search scenarios where **temporal information** is important to satisfy an information need. By temporal information we mean any time reference (e.g., “August 1999”, “last week”, “20th century”, or “January 1 2002”).

Instructions

You're given an incomplete query consisting only of a time reference. Please complete the query by adding the name of a **person, location, or organization** that you think **fits the time reference**

Examples

- Given **December 8, 1980**, you could add *John Lennon*, since he was assassinated on that day
- Given **1789**, you could add *France*, since the French Revolution started in 1789
- Given **July 2012**, you could add *London*, as the hosting city for the 2012 Olympic Games in July and August
- Given **1998**, you could add *Google Inc.*, since the company was founded in that year
- Given **1970s**, you could add *Led Zeppelin*, since the rock band was very successful in that decade
- Given **17th century**, you could add *Isaac Newton*, since he lived in that century

Task

Please complete the following query by adding the name of a **person, location, or organization** that fits the given time reference.

1860

We appreciate your comments (e.g., why you picked this particular person, location, or organization)!

Figure 5.3: Amazon Mechanical Turk HIT to collect queries by letting users fill in an entity that fits a given temporal expression

users' interests and preferences, we ran two user studies on Amazon Mechanical Turk. In our first study, workers were provided with an entity related to one of the topics *Sports, Culture, Technology, or World Affairs* and asked to specify a temporal expression that fits the given entity. In our second study, users were shown a temporal expression corresponding to a *Day, Month, Year, Decade, or Century* and asked to add an entity related to one of the aforementioned topics. Figure 5.3 and Figure 5.4 show screenshots of our HITs. We asked users in both studies to comment on why they chose their particular answer. Examples of comments that we received are:

- boston red sox [october 27, 2004]: *Won 6th World Championship.*
- sewing machine [1850s]: *Isaac Singer invented the sewing machine, then patented the motor for a sewing machine later in that decade.*

Complete a Time-Related Query

We are interested in exploring search scenarios where **temporal information** is important to satisfy an information need. By temporal information we mean any time reference (e.g., “August 1999”, “20th century”, or “January 1 2002”).

Instructions

You're given an incomplete query consisting only of a **person, location, or organization name**. Please complete the query by adding a **time reference** that you think fits the person, location, or organization.

Examples

- Given **barack obama**, you could add *January 20, 2009* as the day of Barack Obama's inauguration
- Given **lehman brothers**, you could add *September 2008* as the month when the company went bankrupt
- Given **beijing**, you could add *2008* as the year when the Olympic Games took place in Beijing
- Given **samuel adams**, you could add *1790s* as the decade when Samuel Adams was Governor of Massachusetts
- Given **red cross**, you could add *1863* as the year when the Red Cross was founded
- Given **germany**, you could add *October 3, 1990* as the day of the German Reunification

Task

Please complete the following query by adding a **time reference** that fits the given **person, location, or organization name**.

boston red sox

We appreciate your comments (e.g., why you picked this particular time reference)!

Figure 5.4: Amazon Mechanical Turk HIT to collect queries by letting users fill in a temporal expression that fits a given entity

- berlin [october 27, 1961]: *Tank standoff at Checkpoint Charlie.*
- chicago bulls [1991]: *The Bulls won the NBA Finals that year.*
- wright brothers [1905]: *The Wright brothers were starting out somewhere around that time.*

Among the queries obtained from our user studies, we selected the 40 queries shown in Figure 5.5. Queries are categorized according to their topic and temporal granularity, giving us a total of 20 query categories, each of which contains two queries.

| | Sports | Culture |
|----------------|-----------------------------------------------------------------------|------------------------------------------------------------------|
| Day | boston red sox [october 27, 2004] ac milan [may 23, 2007] | kurt cobain [april 5, 1994] keith harring [february 16, 1990] |
| Month | stefan edberg [july 1990] italian national soccer team [july 2006] | woodstock [august 1994] pink floyd [march 1973] |
| Year | babe ruth [1921] chicago bulls [1991] | rocky horror picture show [1975] michael jackson [1982] |
| Decade | michael jordan [1990s] new york yankees [1910s] | sound of music [1960s] mickey mouse [1930s] |
| Century | la lakers [21st century] soccer [21st century] | academy award [21st century] jazz music [21st century] |

| | Technology | World Affairs |
|----------------|-------------------------------------------------------------|-------------------------------------------------------------|
| Day | mac os x [march 24, 2001] voyager [september 5, 1977] | berlin [october 27, 1961] george bush [january 18, 2001] |
| Month | thomas edison [december 1891] microsoft halo [june 2000] | poland [december 1970] pearl harbor [december 1941] |
| Year | roentgen [1895] wright brothers [1905] | nixon [1970s] iraq [2001] |
| Decade | internet [1990s] sewing machine [1850s] | vietnam [1960s] monica lewinsky [1990s] |
| Century | musket [16th century] siemens [19th century] | queen victoria [19th century] muhammed [7th century] |

Figure 5.5: Queries categorized according to topic and temporal granularity

Relevance Assessments

Relevance assessments were also collected using Amazon Mechanical Turk. Figure 5.6 shows a screenshot of our HIT. We computed top-10 query results for each query and each method configuration under comparison, pooled them, which yielded a total of 1,251 query-document pairs on the New York Times dataset and 1,220 query-document pairs on the Wikipedia dataset. Each of these query-document pairs was assessed by five workers on Amazon Mechanical Turk. Workers could state whether they considered the document *relevant* or *not relevant* to the query. To prevent spurious assessments, a third option (coined *I don't know*) was provided, which workers should select if they had insufficient information or knowledge to assess the document's relevance. Further, we

Judge the Relevance of a Document to a Query

We are interested in cases where **temporal information** is important to satisfy an information need. By temporal information we mean any time reference (e.g., "August 1999", "last week", "20th century", or "January 1 2002") contained in documents.

Instructions

- **Read** the document (do not just look at the title)
- **Judge** whether the document is relevant or not relevant to the query
- **Explain** your judgment in your own words (i.e., briefly tell us why you think the document is relevant or not relevant)

Tips

- Each document should be judged **on its own merits**, i.e., a document is still relevant even if you've seen other documents containing the same information
- A document is considered relevant if it contains **both textual and temporal information matching the query**
- Only work with **meaningful explanations** will be accepted (i.e., do not just write "relevant" or "not relevant")

Task

Please judge the relevance of the following document to the query **musket 16th century**. Remember, a document is considered relevant if it contains **both textual and temporal information** matching the query.



The screenshot shows a Wikipedia article titled "Pike and shot". The article content is partially visible, starting with "Pike and shot is a historical method of infantry combat, and". A prominent warning box states: "This is an old revision of this page, as edited by Ingolfson (talk | contribs) at 06:18, 4 July 2009. It may differ significantly from the current revision." Navigation links for "Previous revision", "Current revision", and "Newer revision" are visible below the warning box. The Wikipedia logo and navigation menu are also present in the top left of the screenshot.

Please judge the relevance of the above document to the query **musket 16th century** as follows.

Relevant. A relevant document containing both textual and temporal information relevant to the query.

Not relevant. The document is not good because it doesn't contain any relevant information.

I don't know. I don't have enough information to evaluate this document.

Please explain why you think the document is relevant or not relevant!

Figure 5.6: Amazon Mechanical Turk HIT to collect relevance assessments

asked workers to explain in their own words why the document was relevant or not relevant. We found the feedback provided through the explanations extremely insightful. Examples of provided explanations are:

- roentgen [1895]: *Wilhelm Roentgen was alive in 1895 when the building in New York at 150 Nassau Street in downtown Manhattan, NYC was built, they do not ever intersect other than sharing the same timeline of existence for a short while.*
- nixon [1970s]: *This article is relevant. It is a letter to the editor in response to a column about 1970s-era Nixon drug policy.*
- keith harring [february 16, 1990]: *The article does not have any information on Keith Harring, only Laura Harring. Though it contains the keywords Harring and 1990, the article is obviously not what the searcher is looking for.*

Apart from that, when having to explain their assessment, workers seemed more thorough in their assessments. Per completely assessed query-document pair we paid \$0.02 to workers. For the relevance assessments on NYT, workers chose relevant for 33%, not relevant for 63%, and the third option (i.e., *I don't know*) for 4% of the total 6,255 relevance assessments. On WIKI, workers chose relevant for 35%, not relevant for 62%, and the third option (i.e., *I don't know*) for 3% of the total 6,100 relevance assessments. Relevance assessments with the last option are ignored when computing retrieval-effectiveness measures below. To measure the degree of agreement between assessors, we computed the Fleiss' κ statistic [Fle71] as described in Chapter 2. We obtained values of 0.36 and 0.40 on NYT and WIKI, respectively, indicating a fair degree of agreement between assessors.

5.5.2 Experimental Results

We measure the retrieval effectiveness of the methods under comparison using Precision at k ($P@k$) and n DCG at k ($N@k$) as two standard measures described in Chapter 2. When computing $P@k$, we employ majority voting. A document is thus considered relevant to a query if the majority of workers assessed it as relevant. When computing $N@k$, the average relevance grade assigned by workers is determined, interpreting *relevant* as grade 1 and *not relevant* as grade 0.

| | P@5 | N@5 | P@10 | N@10 |
|---------------------------------------------|-------------|-------------|-------------|-------------|
| LM ($\gamma = 0.25$) | 0.33 | 0.34 | 0.30 | 0.32 |
| LM ($\gamma = 0.75$) | 0.38 | 0.39 | 0.37 | 0.38 |
| LMT-IN ($\gamma = 0.25, \lambda = 0.75$) | 0.26 | 0.27 | 0.23 | 0.25 |
| LMT-IN ($\gamma = 0.75, \lambda = 0.75$) | 0.29 | 0.31 | 0.25 | 0.28 |
| LMT-EX ($\gamma = 0.25, \lambda = 0.75$) | 0.36 | 0.36 | 0.32 | 0.33 |
| LMT-EX ($\gamma = 0.5, \lambda = 0.75$) | 0.37 | 0.37 | 0.32 | 0.33 |
| LMTU-IN ($\gamma = 0.25, \lambda = 0.75$) | 0.41 | 0.42 | 0.37 | 0.37 |
| LMTU-IN ($\gamma = 0.75, \lambda = 0.25$) | 0.44 | 0.44 | 0.39 | 0.40 |
| LMTU-EX ($\gamma = 0.25, \lambda = 0.75$) | 0.53 | 0.51 | 0.49 | 0.49 |
| LMTU-EX ($\gamma = 0.5, \lambda = 0.75$) | 0.54 | 0.52 | 0.51 | 0.49 |

Table 5.2: Retrieval effectiveness overall on NYT

| | P@5 | N@5 | P@10 | N@10 |
|---------------------------------------------|-------------|-------------|-------------|-------------|
| LM ($\gamma = 0.25$) | 0.47 | 0.46 | 0.42 | 0.43 |
| LM ($\gamma = 0.75$) | 0.52 | 0.49 | 0.51 | 0.48 |
| LMT-IN ($\gamma = 0.25, \lambda = 0.75$) | 0.39 | 0.37 | 0.29 | 0.31 |
| LMT-IN ($\gamma = 0.75, \lambda = 0.75$) | 0.40 | 0.38 | 0.33 | 0.34 |
| LMT-EX ($\gamma = 0.25, \lambda = 0.75$) | 0.41 | 0.38 | 0.35 | 0.34 |
| LMT-EX ($\gamma = 0.75, \lambda = 0.75$) | 0.43 | 0.39 | 0.36 | 0.36 |
| LMTU-IN ($\gamma = 0.25, \lambda = 0.75$) | 0.50 | 0.48 | 0.44 | 0.43 |
| LMTU-IN ($\gamma = 0.75, \lambda = 0.75$) | 0.54 | 0.50 | 0.48 | 0.46 |
| LMTU-EX ($\gamma = 0.25, \lambda = 0.75$) | 0.57 | 0.51 | 0.54 | 0.50 |
| LMTU-EX ($\gamma = 0.75, \lambda = 0.75$) | 0.60 | 0.53 | 0.56 | 0.51 |

Table 5.3: Retrieval effectiveness overall on WIKI

Overall Retrieval Performance

Table 5.2 and Table 5.3 give retrieval-effectiveness figures computed using all queries and cut-off levels $k = 5$ and $k = 10$ on NYT and WIKI, respectively. For each of the five methods under comparison, the tables show the best-performing and worst-performing configuration with their corresponding values for the mixture parameters γ and λ .

The figures shown support the following observations: (i) on WIKI all methods achieve slightly higher retrieval effectiveness than on NYT, (ii) on both datasets the exclusive mode outperforms the inclusive mode for both LMT and LMTU,

(iii) LMT does not yield an improvement over the baseline LM but even reduces retrieval effectiveness, (iv) LMTU is at par with the baseline LM when the inclusive mode is used and outperforms it significantly when used with the exclusive mode. For LMTU-EX the worst configuration beats the best configuration of the baseline. Further, the worst and best configuration of LMTU-EX are close to each other demonstrating the method’s robustness.

| | Sports | | Culture | | Technology | | World Affairs | |
|---------|-------------|-------------|-------------|-------------|-------------|-------------|---------------|-------------|
| | P@10 | N@10 | P@10 | N@10 | P@10 | N@10 | P@10 | N@10 |
| LM | 0.33 | 0.33 | 0.39 | 0.38 | 0.27 | 0.32 | 0.50 | 0.49 |
| LMT-IN | 0.36 | 0.36 | 0.25 | 0.30 | 0.10 | 0.15 | 0.30 | 0.30 |
| LMT-EX | 0.46 | 0.44 | 0.33 | 0.34 | 0.12 | 0.17 | 0.38 | 0.38 |
| LMTU-IN | 0.46 | 0.44 | 0.41 | 0.42 | 0.21 | 0.27 | 0.48 | 0.48 |
| LMTU-EX | 0.67 | 0.58 | 0.47 | 0.49 | 0.29 | 0.34 | 0.60 | 0.57 |

Table 5.4: Retrieval effectiveness by topic on NYT

| | Sports | | Culture | | Technology | | World Affairs | |
|---------|-------------|-------------|-------------|-------------|-------------|-------------|---------------|-------------|
| | P@10 | N@10 | P@10 | N@10 | P@10 | N@10 | P@10 | N@10 |
| LM | 0.40 | 0.38 | 0.53 | 0.49 | 0.52 | 0.50 | 0.57 | 0.54 |
| LMT-IN | 0.28 | 0.29 | 0.33 | 0.34 | 0.33 | 0.31 | 0.36 | 0.40 |
| LMT-EX | 0.32 | 0.34 | 0.36 | 0.34 | 0.37 | 0.34 | 0.37 | 0.41 |
| LMTU-IN | 0.39 | 0.40 | 0.53 | 0.48 | 0.48 | 0.45 | 0.49 | 0.51 |
| LMTU-EX | 0.54 | 0.48 | 0.57 | 0.51 | 0.61 | 0.54 | 0.52 | 0.52 |

Table 5.5: Retrieval effectiveness by topic on WIKI

Retrieval Performance by Topic

For the best-performing configuration of each method (as given in Table 5.2 and Table 5.3), we compute retrieval-effectiveness measures at cut-off level $k = 10$ and group them by topic.

For NYT the resulting figures are shown in Table 5.4 and support our above observations. Thus, LMTU-EX consistently achieves the highest retrieval effectiveness across all topics. Further, we observe that all methods perform worst on queries from *Technology*. The best performance varies per method and measure.

Table 5.5 shows the resulting figures for WIKI. Here, LMTU-EX performs best on three of the four topics, but achieves retrieval-effectiveness scores slightly lower than those of the baseline LM on queries from *World Affairs*.

| | Day | | Month | | Year | | Decade | | Century | |
|---------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | P@10 | N@10 |
| LM | 0.35 | 0.38 | 0.42 | 0.40 | 0.65 | 0.59 | 0.20 | 0.28 | 0.25 | 0.26 |
| LMT-IN | 0.18 | 0.22 | 0.20 | 0.21 | 0.55 | 0.50 | 0.23 | 0.30 | 0.20 | 0.24 |
| LMT-EX | 0.26 | 0.28 | 0.24 | 0.25 | 0.58 | 0.55 | 0.28 | 0.33 | 0.31 | 0.32 |
| LMTU-IN | 0.33 | 0.36 | 0.47 | 0.46 | 0.59 | 0.56 | 0.34 | 0.35 | 0.24 | 0.27 |
| LMTU-EX | 0.43 | 0.44 | 0.50 | 0.50 | 0.69 | 0.64 | 0.56 | 0.54 | 0.36 | 0.35 |

Table 5.6: Retrieval effectiveness by temporal granularity on NYT

| | Day | | Month | | Year | | Decade | | Century | |
|---------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | P@10 | N@10 |
| LM | 0.35 | 0.37 | 0.55 | 0.49 | 0.75 | 0.63 | 0.50 | 0.50 | 0.38 | 0.39 |
| LMT-IN | 0.11 | 0.17 | 0.10 | 0.16 | 0.66 | 0.60 | 0.50 | 0.47 | 0.25 | 0.28 |
| LMT-EX | 0.11 | 0.18 | 0.10 | 0.16 | 0.70 | 0.60 | 0.55 | 0.51 | 0.31 | 0.33 |
| LMTU-IN | 0.43 | 0.42 | 0.44 | 0.48 | 0.66 | 0.60 | 0.51 | 0.49 | 0.30 | 0.30 |
| LMTU-EX | 0.34 | 0.38 | 0.45 | 0.46 | 0.71 | 0.61 | 0.71 | 0.63 | 0.59 | 0.48 |

Table 5.7: Retrieval effectiveness by temporal granularity on WIKI

Retrieval Performance by Temporal Granularity

In analogy, we group retrieval-effectiveness measurements at cut-off level $k=10$ by temporal granularity – again considering only the best-performing configuration of each method.

Table 5.6 gives the resulting figures for NYT. LMTU-EX consistently achieves the best retrieval performance. Apart from that, we observe significant variations in retrieval effectiveness across temporal granularities for the baseline LM. For queries that include a year, all methods achieve their best performance on NYT. The worst performance varies per method and measure.

For WIKI the resulting figures given in Table 5.7 show a less distinct picture. Thus, for queries containing a month or a year, the baseline LM achieves the best retrieval effectiveness, although LMTU-EX is close behind. LMTU-EX clearly outperforms the baseline LM for queries containing a decade or a century. Interestingly, for queries that contain a day, LMTU-IN achieves the best performance.

Summary

Our experimental evaluation leads us to the following findings. When assessed on the whole of queries, LMTU consistently achieves superior retrieval performance on both datasets. The uncertainty-ignorant LMT model, in contrast, lets retrieval performance deteriorate in comparison to the baseline. For both methods, the exclusive mode of deriving the query from the user’s input performs better than its inclusive counterpart. In summary, (i) considering the uncertainty inherent to temporal expressions is essential and (ii) excluding terms that constitute a temporal expression from the textual part of the query is beneficial.

5.6 Discussion & Outlook

In this work, we have developed a novel approach that integrates temporal expressions seamlessly into a language model retrieval framework, taking into account the uncertainty inherent to temporal expressions. Comprehensive experiments on the New York Times Annotated Corpus and a snapshot of the English Wikipedia, as two publicly-available large-scale document collections, with relevance assessments obtained using Amazon Mechanical Turk showed that our approach substantially improves retrieval effectiveness for temporal information needs.

Outlook

Our focus in this work has been on temporal information needs disclosed by an *explicit* temporal expression in the user’s query.

Often, as somewhat explored in [MJPZ09], queries may not contain such an explicit temporal expression, but still have an associated *implicit* temporal intent. Consider a query such as *bill clinton arkansas* that is likely to allude to Bill Clinton’s time as Governor of Arkansas between 1971 and 1981. Detecting and dealing with such queries is an interesting direction for future research.

Apart from that, temporal information contained in documents may be valuable when trying to provide diverse query results. Even for queries that do not have a temporal intent behind them, the user profits from a result documents that *contain diverse temporal expressions*. Thus, for a query such as *vincent van gogh*, a set of result documents discussing different periods in the famous

painter's life is preferable to a set of documents focused on his final years. Leveraging temporal expressions as a source for result diversification is another interesting direction for future research.

Chapter 6

Conclusions

In this work, we have addressed three problems toward unfolding the full potential of web archives to make them truly valuable resources. In detail:

- We proposed the idea of *time-travel text search* that allows users to search only those portions of a web archive that existed at a time of interest. We proposed the Time-Travel Inverted index (TTIX) as a versatile framework to support time-travel text search. To keep the index compact, we developed temporal coalescing techniques for the types of posting payloads that are needed to support Boolean queries, keyword queries, and phrase queries. Our partitioning strategies allow a fine-tuning of the index with regard to imposed performance requirements or space constraints. Through a comprehensive experimental evaluation on the revision history of the English Wikipedia, a subset of the European Archive, and the New York Times Annotated Corpus, as three representative real-world web archives, we demonstrated the practical viability of our approach to time-travel text search.
- To counter the negative effects that *terminology evolution* has on retrieval effectiveness in web archive search, we introduced a novel query-reformulation technique. By leveraging time-dependent term co-occurrence statistics, our technique is able to identify query reformulations that paraphrase the user's information need using terminology that was prevalent in the past. Using the New York Times Annotated Corpus, as a real-world dataset containing twenty years' worth of newspaper articles, we demonstrated experimentally that our method produces meaningful query reformulations in practice.

- For *temporal information needs* that are satisfied best by documents referring to a particular time, we developed a novel retrieval model that seamlessly integrates temporal expressions into a language modeling approach to information retrieval. Our model makes temporal expressions contained in documents and users' queries first-class citizens, while taking into account their inherent uncertainty. For our experiments on the English Wikipedia and the New York Times Annotated Corpus, we involved real-world users by means of the crowdsourcing platform Amazon Mechanical Turk to collect realistic queries and to gather relevance assessments. Our experiments show that the proposed retrieval model achieves substantial improvements for temporal information needs.

Outlook

The problems addressed in this work are only three among many that arise in the context of web archives. There is thus plenty of opportunity for future research. In the following, we briefly outline some directions that we consider promising and important:

Retrieval Models for Web Archives Our work in Chapter 3 slightly adapted the well-known Okapi BM25 retrieval model for web archives. However, our adaptation does not take into account the fact that there can be many versions of the same relevant document that are almost identical and therefore dilute the query result. Consequently, we consider designing retrieval models specifically for web archives a promising direction of future research. Such retrieval models could, for instance, aggregate the relevance of versions belonging to the same document (as somewhat explored in Berberich et al. [BBW07b]) or aim to find a set of relevant and diverse versions for every result document.

Web Archive Exploration & Mining The techniques presented in this work target users who already have an idea about what they are looking for. Often-times, though, users only know what they are looking for once it is shown to them. Traditionally, data mining and data exploration techniques target these scenarios. Adapting existing techniques or developing novel mining and exploration techniques for web archives is a second promising direction for future

research. For instance, an interesting mining problem, related to terminology evolution considered in Chapter 2, is to efficiently identify terms that changed their meaning in a given time frame. The time axis inherent to web archives opens up interesting problems related to their exploration. One of them would be to identify phrases from a set of document relevant to a query that occur predominantly in documents published during a certain time period; this can be seen as an extension of the techniques for identifying interesting phrases proposed in Simitsis et al. [SBSR08] and Bedathur et al. [BBD⁺10].

Decentralized Web Archiving Nowadays, as discussed in Chapter 2, the majority of web archives is still in the hands of few organizations. Web archives are thus typically organized in a centralized fashion and are thus vulnerable to natural catastrophes (e.g., earthquakes) and not shielded from censorship and manipulation. Peer-to-peer computing addresses the weakness of having a single point of failure by putting the workload on the shoulders of many. Designing and deploying a decentralized web archive is a final promising direction of research that we mention here. Ideally, all aspects of a web archive including acquisition, storage, indexing, and provisioning of web content should be done in a decentralized manner. An initial design for such a decentralized architecture is discussed in Anand et al. [ABB⁺09a]. However, actually implementing and deploying such an architecture is a problem left open. Apart from that, efficient solution to support time-travel text search in a widely-distributed setting would be needed.

Concluding Remarks

Our implementation of the time-travel inverted index has gone through many iterations since the initial related publications in 2007. The most recent version, that was used to conduct the experimental evaluation in Chapter 3, is available for download from the author's homepage or upon e-mail request to the e-mail address given below.

For the experimental evaluation in Chapter 5, temporal expressions were annotated in the English Wikipedia and the New York Annotated Corpus using the TARSQI toolkit, which was a time-consuming endeavor. Furthermore, we collected relevance assessments using Amazon Mechanical Turk. An anonymized

version of the corresponding data is available for download from the author's homepage or upon e-mail request to the e-mail address given below.

<http://www.mpi-inf.mpg.de/~kberberi>

kberberi@mpi-inf.mpg.de

Bibliography

- [AMT] Amazon Mechanical Turk
<http://www.mturk.com>.
- [EA] European Archive
<http://www.europarchive.org>.
- [GTV] Google Timeline View
<http://www.google.com/experimental/>.
- [GWT] Google Web Toolkit
<http://code.google.com/webtoolkit/>.
- [HERITRIX] Heritrix
<http://crawler.archive.org>.
- [IA] Internet Archive.
<http://www.archive.org>.
- [IIPC] The International Internet Preservation Consortium
<http://www.iipc.org>.
- [LIWA] LiWA: Living Web Archives
<http://www.liwa-project.eu>.
- [NYT] New York Times Annotated Corpus
<http://corpus.nytimes.com>.
- [NYTAA] New York Times Article Archive
<http://www.nytimes.com/ref/membercenter/nytarchive.html>.
- [TIMEML] TimeML Specification Language
<http://www.timeml.org>.

- [TIMES] The Times Archive
<http://archive.timesonline.co.uk>.
- [TRE] Text REtrievel Conference
<http://trec.nist.gov>.
- [TSH] TimeSearch History
<http://www.timesearch.info>.
- [WIKI] Wikipedia
<http://www.wikipedia.org>.
- [AAD⁺06] William Y. Arms, Selcuk Aya, Pavel Dmitriev, Blazej J. Kot, Ruth Mitchell, and Lucia Walle. Building a research library for the history of the web. In *JCDL '06: Proceedings of the Joint Conference on Digital Libraries*, pages 95–102, 2006.
- [ABB⁺09a] Avishek Anand, Srikanta Bedathur, Klaus Berberich, Ralf Schenkel, and Christos Tryfonopoulos. EverLast: A distributed architecture for preserving the Web. In *JCDL '09: Proceedings of the Joint Conference on Digital Libraries*, page 331–340., 2009.
- [ABB09b] Irem Arikan, Srikanta Bedathur, and Klaus Berberich. Time Will Tell: Leveraging Temporal Expressions in IR. In *WSDM '09: Late Breaking-Results of the Second ACM International Conference on Web Search and Data Mining*, 2009.
- [ADFW08] Eytan Adar, Mira Dontcheva, James Fogarty, and Daniel S. Weld. Zoetrope: Interacting with the Ephemeral Web. In *UIST '08: Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, pages 239–248, 2008.
- [AF92] Peter G. Anick and Rex A. Flynn. Versioning a Full-text Information Retrieval System. In *SIGIR '92: Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 98–111, 1992.
- [AGBY07] Omar Alonso, Michael Gertz, and Ricardo Baeza-Yates. On the value of temporal information in information retrieval. *SIGIR Forum*, 41(2):35–41, 2007.

-
- [AGMC09] I. Antonellis, H. Garcia-Molina, and C.-C. Chang. Simrank++: Query Rewriting through Link Analysis of the Click Graph. In *Proceedings of the VLDB Endowment*, 1(1):408–421, 2008.
- [All72] Arnold A. Allen. *Probability, Statistics and Queueing Theory with Computer Science Applications*. Academic Press Inc., 1972.
- [All83] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Commun. ACM*, 26(11):832–843, 1983.
- [AM06a] Vo Ngoc Anh and Alistair Moffat. Pruned query evaluation using pre-computed impacts. In *SIGIR '06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 372–379, 2006.
- [AM06b] Vo Ngoc Anh and Alistair Moffat. Pruning strategies for mixed-mode querying. In *CIKM '06: Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 190–197, 2006.
- [AM06c] Vo Ngoc Anh and Alistair Moffat. Structured index organizations for high-throughput text querying. In *SPIRE '06: String Processing and Information Retrieval, 13th International Conference*, pages 304–315, 2006.
- [ARS08] Omar Alonso, Daniel E. Rose, and Benjamin Stewart. Crowdsourcing for relevance evaluation. *SIGIR Forum*, 42(2):9–15, 2008.
- [ATDE09] Eytan Adar, Jaime Teevan, Susan T. Dumais, and Jonathan L. Elsas. The Web Changes Everything: Understanding the Dynamics of Web Content. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 282–291, 2009.
- [AvRdR07] David Ahn, Joris van Rantwijk, and Maarten de Rijke. A cascaded machine learning approach to interpreting temporal expressions. In *HLT-NAACL*, pages 420–427, 2007.
- [BBAW10] Klaus Berberich, Srikanta Bedathur, Omar Alonso, and Gerhard Weikum. A Language Modeling Approach for Temporal Informa-

- tion Needs. In *ECIR '10: Advances in Information Retrieval, 32nd European Conference on IR Research*, 2010.
- [BBD⁺10] Srikanta Bedathur, Klaus Berberich, Jens Dittrich, Nikos Mamoulis, and Gerhard Weikum. Interesting-Phrase Mining for Ad-Hoc Text Analytics. In *Proceedings of the VLDB Endowment*, 2010.
- [BBNW07a] Klaus Berberich, Srikanta Bedathur, Thomas Neumann, and Gerhard Weikum. FluxCapacitor: Efficient Time-Travel Text Search. In *VLDB '07: Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 1414–1417, 2007.
- [BBNW07b] Klaus Berberich, Srikanta Bedathur, Thomas Neumann, and Gerhard Weikum. A time machine for text search. In *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 519–526, 2007.
- [BBSW09] Klaus Berberich, Srikanta Bedathur, Mauro Sozio, and Gerhard Weikum. Bridging the Terminology Gap in Web Archive Search. In *WebDB '09: 12th International Workshop on the Web and Databases*, 2009.
- [BBVW06] Klaus Berberich, Srikanta Bedathur, Michalis Vazirgiannis, and Gerhard Weikum. BuzzRank...and the Trend is Your Friend. In *WWW '06: Proceedings of the 15th International Conference on World Wide Web*, pages 937–938, 2006.
- [BBW07a] Klaus Berberich, Srikanta Bedathur, and Gerhard Weikum. A Pocket Guide to Web History. In *SPIRE '07: String Processing and Information Retrieval, 14th International Conference*, pages 86–97, 2007.
- [BBW07b] Klaus Berberich, Srikanta Bedathur, and Gerhard Weikum. Efficient Time-Travel on Versioned Text Collections. In *Fachtagung für Datenbanken in Business, Wirtschaft und Technik*, 2007.

- [BBW08] Klaus Berberich, Srikanta Bedathur, and Gerhard Weikum. Tunable Word-Level Index Compression for Versioned Corpora. In *EIIR '08: Workshop on Efficiency Issues in Information Retrieval*, 2008.
- [BBWV07] Klaus Berberich, Srikanta Bedathur, Gerhard Weikum, and Michalis Vazirgiannis. Comparing apples and oranges: normalized pagerank for evolving graphs. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 1145–1146, 2007.
- [BBS10] Andreas Broschart, Klaus Berberich, and Ralf Schenkel. Evaluating the Potential of Explicit Phrases on Retrieval Quality. In *ECIR '10: Advances in Information Retrieval, 32nd European Conference on IR Research*, 2010.
- [BCL06] Stefan Büttcher, Charles L. A. Clarke, and Brad Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In *SIGIR '06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 621–622, 2006.
- [BDFL07] Lou Burnard, Milena Dobрева, Norbert Fuhr, and Anke Lüdeling, editors. *Digital Historical Corpora - Architecture, Annotation, and Retrieval, 03.12. - 08.12.2006*, Volume 06491 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.
- [BEF⁺06] Andrei Z. Broder, Nadav Eiron, Marcus Fontoura, Michael Herscovici, Ronny Lempel, John McPherson, Runping Qi, and Eugene J. Shekita. Indexing Shared Content in Information Retrieval Systems. In *EDBT '06: Advances in Database Technology, 10th International Conference on Extending Database Technology*, pages 313–330, 2006.
- [BF04] Nicola Bertoldi and Marcello Federico. Statistical models for monolingual and bilingual information retrieval. *Inf. Retr.*, 7(1-2):53–72, 2004.
- [BG09] Delphine Bernhard and Iryna Gurevych. Combining Lexical Semantic Resources with Question & Answer Archives for

- Translation-Based Answer Finding. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics*, pages 728–736, 2009.
- [BGO⁺96] Bruno Becker, Stephan Gschwind, Thomas Ohler, Bernhard Seeger, and Peter Widmayer. An asymptotically optimal multiversion b-tree. *The VLDB Journal*, 5(4):264–275, 1996.
- [BH99] Michael Burrows and Andrew L. Hisgen. Method and apparatus for generating and searching range-based index of word locations. U.S. Patent 5,915,251, 1999.
- [BL85] Chris Buckley and Alan F. Lewit. Optimization of inverted vector searches. In *SIGIR '85: Proceedings of the 8th Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 97–110, 1985.
- [BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998.
- [BS96] Jochen Van den Bercken and Bernhard Seeger. Query processing techniques for multiversion access methods. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 168–179, 1996.
- [BS07] Pierre-Yves Berger and Jacques Savoy. Selecting automatically the best query translations. In *RIAO '07: Computer-Assisted Information Retrieval (Recherche d'Information et ses Applications)*, 2007.
- [BSS96] Michael H. Böhlen, Richard Thomas Snodgrass, and Michael D. Soo. Coalescing in temporal databases. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 180–191, 1996.
- [BSV05] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. Paradoxical effects in pagerank incremental computations. *Internet Mathematics*, 2(3), 2005.

- [BSWZ03] Bodo Billerbeck, Falk Scholer, Hugh E. Williams, and Justin Zobel. Query expansion using associated queries. In *CIKM '03: Proceedings of the Twelfth International Conference on Information and Knowledge Management*, pages 2–9, 2003.
- [BVW05] Klaus Berberich, Michalis Vazirgiannis, and Gerhard Weikum. Time-aware Authority Ranking. *Internet Mathematics*, 2(3):301–332, 2005.
- [BY05] Ricardo A. Baeza-Yates. Searching the future. In *Proceedings of ACM SIGIR Workshop MF/IR*, 2005.
- [BYRN99] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [CCF⁺01] David Carmel, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, Yoelle S. Maarek, and Aya Soffer. Static index pruning for information retrieval systems. In *SIGIR '01: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–50, 2001.
- [CDG⁺07] Carlos Castillo, Debora Donato, Aristides Gionis, Vanessa Murdock, and Fabrizio Silvestri. Know your neighbors: web spam detection using the web topology. In *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 423–430, 2007.
- [CGR05] Gianna M. Del Corso, Antonio Gulli, and Francesco Romani. Ranking a stream of news. In *WWW '05: Proceedings of the 14th International Conference on World Wide Web*, pages 97–106, 2005.
- [Cha86] Bernard Chazelle. Filtering search: a new approach to query answering. *SIAM J. Comput.*, 15(3):703–724, 1986.
- [Cha02] Soumen Chakrabarti. *Mining the Web*. Morgan Kaufmann Publishers, 2002.
- [CMS09] W. Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines Information Retrieval in Practice*. Addison Wesley, 2009.

- [Com79] Douglas Comer. The Ubiquitous B-Tree. *ACM Comput. Surv.*, 11(2):121–137, 1979.
- [CP08] Matthew Chang and Chung Keung Poon. Efficient phrase querying with common phrase index. *Inf. Process. Manage.*, 44(2):756–769, 2008.
- [CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.
- [CTC05] Kevyn Collins-Thompson and Jamie Callan. Query expansion using random walk models. In *CIKM '05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, pages 704–711, 2005.
- [CTL91] W. Bruce Croft, Howard R. Turtle, and David D. Lewis. The use of phrases and structured queries in information retrieval. In *SIGIR '91: Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 32–45, 1991.
- [CV07] Rudi L. Cilibrasi and Paul M. B. Vitanyi. The google similarity distance. *IEEE Trans. on Knowl. and Data Eng.*, 19(3):370–383, 2007.
- [CWNM02] Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. Probabilistic query expansion using query logs. In *WWW '02: Proceedings of the 11th International Conference on World Wide Web*, pages 325–332, 2002.
- [dBCvKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2008.
- [DG10] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, 2010.
- [DGI08] Wisam Dakka, Luis Gravano, and Panagiotis G. Ipeirotis. Answering general time sensitive queries. In *CIKM '08: Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 1437–1438, 2008.

- [dJRH05] Franciska de Jong, Henning Rode, and Djoerd Hiemstra. Temporal language models for the disclosure of historical text. In *Humanities, computers and cultural heritage: Proceedings of the XVIth International Conference of the Association for History and Computing (AHC 2005)*, pages 161–168, 2005.
- [EGF06] Andrea Ernst-Gerlach and Norbert Fuhr. Generating search term variants for text collections with historic spellings. In *ECIR '06: Advances in Information Retrieval, 32nd European Conference on IR Research*, pages 49–60, 2006.
- [EGF07] Andrea Ernst-Gerlach and Norbert Fuhr. Retrieval in text collections with historic spelling using linguistic and spelling variants. In *JCDL '07: Proceedings of the Joint Conference on Digital Libraries*, pages 333–341, 2007.
- [FB02] Marcello Federico and Nicola Bertoldi. Statistical cross-language information retrieval using n-best query translations. In *SIGIR '02: Proceedings of the 25th annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 167–174, 2002.
- [Fle71] Joseph L. Fleiss. Measuring nominal scale agreement among many raters. In *Psychological Bulletin*, volume 76, pages 323–327, 1971.
- [FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.
- [FLQZ07] Marcus Fontoura, Ronny Lempel, Runping Qi, and Jason Zien. Inverted index support for numeric search. *Internet Mathematics*, 3(2):153–185, 2007.
- [FMNW04] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L. Wiener. A large-scale study of the evolution of web pages. *Software: Practice and Experience*, 34(2):213–237, 2004.

- [Hav02] Taher H. Haveliwala. Topic-sensitive PageRank. In *WWW '02: Proceedings of the Eleventh International Conference on World Wide Web*, pages 517–526, 2002.
- [HCB⁺08] Rong Hu, Weizhu Chen, Peng Bai, Yansheng Lu, Zheng Chen, and Qiang Yang. Web query translation via web log mining. In *SIGIR '08: Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 749–750, 2008.
- [HLY07] Michael Herscovici, Ronny Lempel, and Sivan Yogev. Efficient indexing of versioned document sequences. In *ECIR '07: Advances in Information Retrieval, 33rd European Conference on IR Research*, pages 76–87, 2007.
- [HS86] W. Daniel Hillis and Guy L. Steele, Jr. Data parallel algorithms. *Commun. ACM*, 29(12):1170–1183, 1986.
- [HYS09] Jinru He, Hao Yan, and Torsten Suel. Compact full-text indexing of versioned document collections. In *CIKM '09: Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 415–424, 2009.
- [IP95] Yannis E. Ioannidis and Viswanath Poosala. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 233–244, 1995.
- [JD07] Rosie Jones and Fernando Diaz. Temporal profiles of queries. *ACM Trans. Inf. Syst.*, 25(3):14, 2007.
- [JDB⁺97] Christian S. Jensen, Curtis E. Dyreson, Michael H. Böhlen, James Clifford, Ramez Elmasri, Shashi K. Gadia, Fabio Grandi, Patrick J. Hayes, Sushil Jajodia, Wolfgang Käfer, Nick Kline, Nikos A. Lorentzos, Yannis G. Mitsopoulos, Angelo Montanari, Daniel A. Nonen, Elisa Peressi, Barbara Pernici, John F. Roddick, Nandlal L. Sarda, Maria Rita Scalas, Arie Segev, Richard T. Snodgrass, Michael D. Soo, Abdullah Uz Tansel, Paolo Tiberio, and Gio

- Wiederhold. The consensus glossary of temporal database concepts. In *Temporal Databases: Research and Practice.*, pages 367–405, 1997.
- [JK02] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.
- [JKM⁺98] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *VLDB '98: Proceedings of the 24th International Conference on Very Large Data Bases*, pages 275–286, 1998.
- [JRMG06] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating Query Substitutions. In *WWW '06: Proceedings of the 15th International Conference on World Wide Web*, pages 387–396, 2006.
- [JWR00a] Karen Sparck Jones, Steve Walker, and Stephen E. Robertson. A probabilistic model of information retrieval: development and comparative experiments - part 1. *Inf. Process. Manage.*, 36(6):779–808, 2000.
- [JWR00b] Karen Sparck Jones, Steve Walker, and Stephen E. Robertson. A probabilistic model of information retrieval: development and comparative experiments - part 2. *Inf. Process. Manage.*, 36(6):809–840, 2000.
- [KAKdR06] Marijn Koolen, Frans Adriaans, Jaap Kamps, and Maarten de Rijke. A cross-language approach to historic document retrieval. In *ECIR '06: Advances in Information Retrieval, 32nd European Conference on IR Research*, pages 407–419, 2006.
- [KB00] Douglas B. Koen and Walter Bender. Time frames: Temporal augmentation of the news. *IBM Systems Journal*, 39(3&4):597–616, 2000.
- [KC05] Pawel Jan Kalczynski and Amy Chou. Temporal document retrieval model for business news archives. *Inf. Process. Manage.*, 41(3):635–650, 2005.

- [KCHP01] Eamonn J. Keogh, Selina Chu, David Hart, and Michael J. Pazzani. An online algorithm for segmenting time series. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 289–296, 2001.
- [KJV83] Scott Kirkpatrick, D. Gelatt Jr., and Mario P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.
- [Kle99] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.
- [KN08] Nattiya Kanhabua and Kjetil Nørvåg. Improving temporal language models for determining time of non-timestamped documents. In *ECDL '08: Proceedings of the 12th European conference on Research and Advanced Technology for Digital Libraries*, pages 358–370, 2008.
- [KRVV96] Paris Kanellakis, Sridhar Ramaswamy, Darren E. Vengroff, and Jeffrey Scott Vitter. Indexing for data models with constraints and classes. *J. Comput. Syst. Sci.*, 52(3):589–612, 1996.
- [KT05] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
- [K73] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.
- [LC03] Xiaoyan Li and W. Bruce Croft. Time-based language models. In *CIKM '03: Proceedings of the Twelfth International Conference on Information and Knowledge Management*, pages 469–475, 2003.
- [LHNZ08] David Lomet, Mingsheng Hong, Rimma Nehme, and Rui Zhang. Transaction time indexing with version compression. *Proceedings of the VLDB Endowment*, 1(1):870–881, 2008.
- [LJC05] Yi Liu, Rong Jin, and Joyce Y. Chai. A maximum coherence model for dictionary-based cross-language information retrieval. In *SIGIR '05: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 536–543, 2005.

- [LS89] David Lomet and Betty Salzberg. Access methods for multiversion data. In *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 315–324, 1989.
- [Mas06] Julien Masanès. *Web Archiving*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [MBSC97] Mandar Mitra, Chris Buckley, Amit Singhal, and Claire Cardie. An Analysis of Statistical and Syntactic Phrases. In *RIAO '97: Computer-Assisted Information Retrieval (Recherche d'Information et ses Applications)*, pages 200–217, 1997.
- [MC91] George A. Miller and Walter G. Charles. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28, 1991.
- [MD05] Christof Monz and Bonnie J. Dorr. Iterative translation disambiguation for cross-language information retrieval. In *SIGIR '05: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 520–527, 2005.
- [MJPZ09] Donald Metzler, Rosie Jones, Fuchun Peng, and Ruiqiang Zhang. Improving search relevance for implicitly temporal queries. In *SIGIR '09: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 700–701, 2009.
- [MOPW00] Peter Muth, Patrick O'Neil, Achim Pick, and Gerhard Weikum. The LHAM log-structured history data access method. *The VLDB Journal*, 8(3-4):199–221, 2000.
- [MPG05] Inderjeet Mani, James Pustejovsky, and Rob Gaizauskas. *The Language of Time: A Reader*. Oxford University Press, 2005.
- [MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

- [MS99] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.
- [MW00] Inderjeet Mani and George Wilson. Robust temporal processing of news. In *ACL '00: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 69–76, 2000.
- [MZ96] Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.*, 14(4):349–379, 1996.
- [MZC08] Qiaozhou Mei, Dengyong Zhou, and Kenneth Church. Query Suggestion using Hitting Time. In *CIKM '08: Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 469–478, 2008.
- [NC07] Alexandros Ntoulas and Junghoo Cho. Pruning policies for two-tiered inverted index with correctness guarantee. In *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 191–198, 2007.
- [NCO04] Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. What's New on the Web?: The Evolution of the Web from a Search Engine Perspective. In *WWW '04: Proceedings of the 13th Conference on World Wide Web*, pages 1–12, 2004.
- [NN06] Kjetil Nørvåg and Albert Overskeid N Nybø. Dyst: Dynamic and Scalable Temporal Text Indexing. In *TIME '06: 13th International Symposium on Temporal Representation and Reasoning*, 0:204–211, 2006.
- [Nør03] Kjetil Nørvåg. Space-efficient support for temporal text indexing in a document archive context. In *ECDL '03: Research and Advanced Technology for Digital Libraries, 7th European Conference*, pages 511–522, 2003.
- [Nør04] Kjetil Nørvåg. Supporting temporal text-containment queries in temporal document databases. *Data Knowl. Eng.*, 49(1):105–125, 2004.

- [NRD08] Sérgio Nunes, Cristina Ribeiro, and Gabriel David. Use of Temporal Expressions in Web Search. In *ECIR '08: Advances in Information Retrieval, 30th European Conference on IR Research*, pages 580–584, 2008.
- [PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.
- [PC98] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information retrieval*, pages 275–281, 1998.
- [PCI⁺03] James Pustejovsky, José M. Castaño, Robert Ingria, Roser Sauri, Robert J. Gaizauskas, Andrea Setzer, Graham Katz, and Dragomir R. Radev. TimeML: Robust specification of event and temporal expressions in text. In *New Directions in Question Answering*, pages 28–34, 2003.
- [QF93] Yonggang Qiu and Hans-Peter Frei. Concept based query expansion. In *SIGIR '93: Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–169, 1993.
- [Rab90] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Readings in speech recognition*, pages 267–296, 1990.
- [RG03] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, Inc., 2003.
- [Ram97] Sridhar Ramaswamy. Efficient indexing for constraint and temporal databases. In *ICDT '97: Proceedings of the 6th International Conference on Database Theory*, pages 419–431, 1997.
- [RG65] Herbert Rubenstein and John B. Goodenough. Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633, 1965.

- [RN03] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.
- [S00] Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann Publishers Inc., 2000.
- [SBSR08] Alkis Simitsis, Akanksha Baid, Yannis Sismanis, and Berthold Reinwald. Multidimensional Content eXploration. In *VLDB '08: Proceedings of the 34rd International Conference on Very Large Data Bases*, pages 660–671, 2008.
- [SBwH⁺07] Ralf Schenkel, Andreas Broschart, Seung won Hwang, Martin Theobald, and Gerhard Weikum. Efficient text proximity search. In *SPIRE '07: String Processing and Information Retrieval, 14th International Conference*, pages 287–299, 2007.
- [SH91] Frank K. Soong and Eng-Fong Huang. A tree-trellis based fast search for finding the n-best sentence hypotheses in continuous speech recognition. In *ICASSP '91: Proceedings of the Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference*, pages 705–708, 1991.
- [SH06] Mehran Sahami and Timothy D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW '06: Proceedings of the 15th International Conference on World Wide Web*, pages 377–386, 2006.
- [ST99] Betty Salzberg and Vassilis J. Tsotras. Comparison of access methods for time-evolving data. *ACM Comput. Surv.*, 31(2):158–221, 1999.
- [Sta06] Michael Stack. Full Text Search of Web Archive Collections. In *IWAW '06: Proceedings of the 6th International Workshop on Web Archiving*, 2006.
- [SWY75] Gerard Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.
- [TCG⁺93] Abdullah Uz Tansel, James Clifford, Shashi K. Gadia, Sushil Jajodia, Arie Segev, and Richard T. Snodgrass, editors. *Tem-*

- poral Databases: Theory, Design, and Implementation.* Benjamin/Cummings, 1993.
- [TF95] Howard Turtle and James Flood. Query evaluation: strategies and optimizations. *Inf. Process. Manage.*, 31(6):831–850, 1995.
- [TIR⁺08] Nina Tahmasebi, Tereza Iofciu, Thomas Risse, Claudia Nieder'ee, and Wolf Siberski. Terminology evolution in web archiving: Open issues. In *IWAW '08: Proceedings of the 8th International Workshop on Web Archiving*, 2008.
- [TSW05] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. Efficient and self-tuning incremental query expansion for top-k query processing. In *SIGIR '05: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 242–249, 2005.
- [TT06] Evimaria Terzi and Panayiotis Tsaparas. Efficient Algorithms for Sequence Segmentation. In *SIAM Data Mining Conference*, 2006.
- [TZ07] Tao Tao and ChengXiang Zhai. An exploration of proximity measures in information retrieval. In *SIGIR '07: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 295–302, 2007.
- [VM09] Marc Verhagen and Jessica L. Moszkowicz. Temporal Annotation and Representation. In *Language and Linguistics Compass*, volume 3, pages 517–536, 2009.
- [VMS⁺05] Marc Verhagen, Inderjeet Mani, Roser Sauri, Jessica Littman, Robert Knippen, Seok Bae Jang, Anna Rumshisky, John Phillips, and James Pustejovsky. Automating Temporal Annotation with TARSQI. In *ACL*, 2005.
- [VRJ03] Olga Vechtomova, Stephen Robertson, and Susan Jones. Query expansion with long-span collocates. *Inf. Retr.*, 6(2):251–273, 2003.
- [WMB99] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing gigabytes (2nd ed.): compressing and indexing documents and images.* Morgan Kaufmann publishers Inc., San Francisco, CA, USA, 1999.

- [WZB04] Hugh E. Williams, Justin Zobel, and Dirk Bahle. Fast phrase querying with combined indexes. *ACM Trans. Inf. Syst.*, 22(4):573–594, 2004.
- [XC96] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. In *SIGIR '96: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 4–11, 1996.
- [ZL04] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.
- [ZM06] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006.
- [ZR02] Ingrid Zukerman and Bhavani Raskutti. Lexical query paraphrasing for document retrieval. In *COLING*, 2002.
- [ZRW02] Ingrid Zukerman, Bhavani Raskutti, and Yingying Wen. Experiments in query paraphrasing for information retrieval. In *Australian Joint Conference on Artificial Intelligence*, pages 24–35, 2002.
- [ZS07] Jiangong Zhang and Torsten Suel. Efficient search in large textual collections with redundancy. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 411–420, 2007.
- [ZSLW09] Mingjie Zhu, Shuming Shi, Mingjing Li, and Ji-Rong Wen. Effective top-k computation with term-proximity support. *Inf. Process. Manage.*, 45(4):401–412, 2009.
- [ZSYW08] Qi Zhang, Fabian M. Suchanek, Lihua Yue, and Gerhard Weikum. TOB: Timely Ontologies for Business Relations. In *WebDB '08: 11th International Workshop on the Web and Databases*, 2008.
- [ZY09] Jinglei Zhao and Yeogirl Yun. A proximity language model for information retrieval. In *SIGIR '09: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 291–298, 2009.

Appendix A

Query Workloads

deaths in 2006, hurricane katrina, playboy, gwen stefani, ask jeeves, randy orton, darfur, wiki, johnny cash, goggle, sudoku, wwe, pancreas, cocaine, scientology, vagina, cold war, vietnam war, ash wednesday, hanson foundation, columbine, pete wentz, shakira, french revolution, world war ii, may day, good friday, elmo s world, american idol, wikipedia, david blaine, the last supper, incest, metaphor, truman capote, mozart, whitney houston, kkk, industrial revolution, tsunami, 69, opus dei, led zeppelin, brazil, alexander the great, june carter, agnostic, lymph nodes, appendix, wiccan, penthouse, marcheline bertrand, haiku, mexico, rosa parks, hotmail, germany, x men, prohibition, john lennon, bestiality, chuck norris jokes, africa, spain, sodomy, dopamine, acre, cuba, genocide, roe v wade, mortal kombat characters, tuberculosis, morphine, domino harvey, existentialism, wikipedia, palm sunday, debra lafave, potassium, spleen, sociopath, priory of sion, serotonin, karma, wiki pedia, gloria vanderbilt, high school musical 2, adolf hitler, trees, italy, the beatles, fidel castro, andy milonakis, randy jackson, maya angelou, limewire, henry ford, john adams, june carter cash, stalin, buddha, the cold war, romanticism, mrsa, abraham lincoln, rome, emo, nudity, watergate, manhunt net, kama sutra, bees, anderson cooper, julius caesar, encarta, dna, dominican republic, manifest destiny, hawaii, truman doctrine, korean war, concentration camps, camels, liver, abortion, harlem renaissance, dubai, audie murphy, beethoven, facebook, oxycodone, the great depression, george w bush, pus, charmed, vatican city, charles darwin, dana reeve, united 93, imperialism, robert e lee, polygamy, the american dream, marijuana, candy samples, the da vinci code, snopes, renaissance, cher, sportsnet new york

Figure A.1: Boolean/Keyword queries used for WIKI dataset

inheritance tax, euro, queen elizabeth, london underground, statistics, criminal records, prince william, national archives, public records, princess diana, royal family, tony blair, windsor castle, prince harry, elizabeth i, wills, scotland, childcare, buckingham palace, london, house of windsor, prince philip, king henry viii, child support, william the conqueror, probate, edinburgh castle, queen elizabeth ii, queen elizabeth i, family history, united kingdom, minimum wage, social security, queen victoria, dfes, maternity leave, prince charles, northumberland, fsa, dress up games, criminal record, traffic signs, pensions, thames barrier, bury st edmunds, british monarchy, uk patent office, mary queen of scots, disability, british crown jewels, doh, dwp, mi6, union flag, durham, default judgement, union jack, drugs, detr, avalanches, risk assessment, limited liability partnership, woodchurch, lady rose windsor, immigration acts, diana princess of wales, st james park, animal by products, asbestos, bird flu, jenny nimmo, british passport, family tree, millenium development goals, dsa, historic scotland, mi5, strychnine, satelite, kensington palace, inclusion, tartan day, benzene in soft drinks, breast implants, teen pregnancy, british royal family, british royalty, farming, buses, queen of england, london metro, child care, earl of lichfield, employers, jobseekers, city of liverpool, paul howes, social security gov, cardiff, cotswold, gerald walker, mexicans in the united kingdom, moderate mental retardation, highgrove, land pollution, general register office for scotland, preparing for emergencies, cyndi s list, social services, uk national archives, prince andrew, angel of the north, asset recovery, mind body and soul, scottish coat of arms, london congestion charge, kensington, the palace of holyroodhouse, uk driving licence, iwerne courtney, princess margaret, medicines, hampshire england, stillbirths, anti social behaviour, core training, belfast northern ireland, teaching assistant, hedgehogs, king edward, charles, collage, good friday, gateway, westminster, ireland, mary slessor, margaret thatcher, prisons in scotland, direct grant, camp hill prison, policy making, toyota city japan, portsmouth, queen, inland revenue, noise pollution, balmoral castle, the tudors, prince

Figure A.2: Boolean/Keyword queries used for UKGOV dataset

washington post, weather, cnn news, washingtonpost, ftd, lou dobbs, flowers, time magazine, msnbc, wall street journal, sports illustrated, larry king, cnnmoney, nancy grace, cnn money, horoscopes, news, political cartoons, funny videos, johnny cash, cnnsi, war in iraq, larry king live, anderson cooper, decks, princess diana, time, the washington post, cartoons, redskins, florists, fortune magazine, cw network, world news, gmt, miss manners, patios, money magazine, don knotts, pope john paul ii, stocks, money, oil prices, retirement calculator, jackie robinson, chris penn, enron, fingernails, dixie chicks, fortune 500, dana reeve, videos, washington redskins, the wall street journal, crosswords, lou dobbs tonight, opus dei, september 11 2001, numerology, dana reeves, wnyt, london weather, washington dc metro, stock quote, drywall, headline news, interest rates, mutual funds, cnfn, weed, sports, sudoku, storm shutters, sat scores, greenwich mean time, hurricane katrina, camcorders, time zones, preteen models, bathrooms, cold war, eleanor roosevelt, florist, send flowers, e pierce marshall, iraq war, laptops, child safety, mother teresa, retaining walls, hurricane shutters, scanners, fortune, drive time, luther vandross, cars, sandra dee, world time, cost of living, aids in africa, henry ford, playground equipment, greenwich time, william kennedy smith, nfl, colonics, patrick kennedy, norma mccorvey, bed bath and beyond, rosa parks, drive-time, girls gone wild, political cartoon, soundboards, lifetime products, fica, dubey, zulu time, o j simpson, katharine graham, chandra levy, seven wonders of the world, nba, barbaro, dennis weaver, civil rights movement, video, hometime, bill cosby, twin towers, lucille ball, home improvements, watergate, dc metro, pheromones, kimberly dozier, wall street, chore charts, local news, immigration bill, health, bruising, thrush, dudley moore, chanel sunglasses, retirement, flight 93, bowflex, deal or no deal, john ritter

Figure A.3: Boolean/Keyword queries used for NYT dataset

deaths in 2006, free encyclopedia, hurricane katrina, gwen stefani, ask jeeves, randy orton, johnny cash, cold war, vietnam war, ash wednesday, hanson foundation, pete wentz, french revolution, world war ii, may day, good friday, elmo s world, american idol, david blaine, the last supper, truman capote, whitney houston, industrial revolution, opus dei, led zeppelin, alexander the great, june carter, lymph nodes, marcheline bertrand, rosa parks, x men, john lennon, chuck norris jokes, roe v wade, mortal kombat characters, domino harvey, palm sunday, debra lafave, priory of sion, gloria vanderbilt, high school musical 2, adolf hitler, the beatles, fidel castro, andy milonakis, randy jackson, maya angelou, henry ford, john adams, june carter cash, the cold war, abraham lincoln, manhunt net, kama sutra, anderson cooper, julius caesar, dominican republic, manifest destiny, truman doctrine, korean war, concentration camps, harlem renaissance, audie murphy, the great depression, george w bush, vatican city, charles darwin, dana reeve, united 93, robert e lee, the american dream, candy samples, the da vinci code, sportsnet new york, stephanie mcmahon, thomas edison, winston churchill, cleveland steamer, cinco de mayo, spanish inquisition, donnie mcclurkin, emily rose, da vinci code, statue of liberty, mississippi river, timothy treadwell, florence nightingale, gospel of judas, chris penn, ku klux klan, hurricane rita, mayo clinic, neil armstrong, deal or no deal, panama canal, zac efron, julian beever, scientific method, maslow s hierarchy of needs, knights templar, dixie chicks, john d rockefeller, flavor flav, john f kennedy, crystal meth, emancipation proclamation, pamela rogers, joan of arc, andrea lowell, roman numerals, sex position, chris daughtry, spiderman 3, silent hill, hurricane wilma, monroe doctrine, sonny moore, peter tomarken, guns n roses, simon cowell, tet offensive, yalta conference, martin luther, bill gates, eleanor roosevelt, jack dunphy, blood tests, charlie rose, xiaolin showdown, louisiana purchase, george washington, kelly clarkson, memorial day, vivian liberto, stadium arcadium, george rr martin, drudge report, 2006 hurricane season, edward r murrow, john jacob astor, ronald reagan, gnarls barkley, skull island, tiffany fallon, peter wentz, eminent domain, intelligent design, global warming, v for vendetta, nicole scherzinger

Figure A.4: Phrase queries used for WIKI dataset

inheritance tax, queen elizabeth, london underground, criminal records, prince william, national archives, public records, princess diana, royal family, tony blair, windsor castle, prince harry, elizabeth i, buckingham palace, house of windsor, prince philip, king henry viii, child support, william the conqueror, edinburgh castle, queen elizabeth ii, queen elizabeth i, family history, united kingdom, minimum wage, social security, queen victoria, maternity leave, prince charles, dress up games, criminal record, traffic signs, thames barrier, bury st edmunds, british monarchy, uk patent office, mary queen of scots, british crown jewels, union flag, default judgement, union jack, risk assessment, limited liability partnership, lady rose windsor, immigration acts, diana princess of wales, st james park, animal by products, bird flu, jenny nimmo, british passport, family tree, millenium development goals, historic scotland, kensington palace, tartan day, benzene in soft drinks, breast implants, teen pregnancy, british royal family, british royalty, queen of england, london metro, child care, earl of lichfield, city of liverpool, paul howes, social security gov, gerald walker, mexicans in the united kingdom, moderate mental retardation, land pollution, general register office for scotland, preparing for emergencies, cyndi s list, social services, uk national archives, prince andrew, angel of the north, asset recovery, mind body and soul, scottish coat of arms, london congestion charge, the palace of holyroodhouse, uk driving licence, iwerne courtney, princess margaret, hampshire england, anti social behaviour, core training, belfast northern ireland, teaching assistant, king edward, good friday, mary slessor, margaret thatcher, prisons in scotland, direct grant, camp hill prison, policy making, toyota city japan, inland revenue, noise pollution, balmoral castle, the tudors, protection from abuse, yin yang symbol, prince of wales, loan sharks, vitamin k, driving test, foreign office, the royal family, parental responsibility, prince phillip, prince edward, english monarchy, uk passport, british government, uk immigration, birth certificates, duke of windsor, road surfaces, summary judgement, civil service, victorian houses, brain stem death, stuart kings, southampton england, loan shark, healthcare in denmark, miscue analysis, involuntary manslaughter, king charles ii, probate court, northern ireland, Acrobat Reader, sexual acts, key man insurance, witness intimidation

Figure A.5: Phrase queries used for UKGOV dataset

washington post, cnn news, lou dobbs, time magazine, wall street journal, sports illustrated, larry king, nancy grace, cnn money, political cartoons, funny videos, johnny cash, war in iraq, larry king live, anderson cooper, princess diana, the washington post, cnn headline news, fortune magazine, cw network, world news, miss manners, money magazine, don knotts, pope john paul ii, oil prices, retirement calculator, jackie robinson, chris penn, dixie chicks, fortune 500, dana reeve, washington redskins, the wall street journal, lou dobbs tonight, opus dei, september 11 2001, dana reeves, london weather, washington dc metro, stock quote, headline news, interest rates, mutual funds, storm shutters, sat scores, greenwich mean time, hurricane katrina, time zones, pre-teen models, cold war, eleanor roosevelt, send flowers, e pierce marshall, iraq war, child safety, mother teresa, retaining walls, hurricane shutters, drive time, luther vandross, sandra dee, world time, cost of living, aids in africa, henry ford, playground equipment, greenwich time, william kennedy smith, patrick kennedy, norma mccorvey, bed bath and beyond, rosa parks, girls gone wild, political cartoon, lifetime products, zulu time, o j simpson, katharine graham, chandra levy, seven wonders of the world, dennis weaver, civil rights movement, bill cosby, twin towers, lucille ball, home improvements, dc metro, kimberly dozier, wall street, chore charts, local news, immigration bill, dudley moore, chanel sunglasses, flight 93, deal or no deal, john ritter, rolex watches, dow jones, sarah hughes, the war in iraq, robert palmer, oklahoma city bombing, gay boys, september 11, david bloom, captain kangaroo, time inc, ethernet card, funny video, larry king show, dubey schaldenbrand, cnn international, stem cell research, kidney infection, current events, 9 11 01, world trade center tenants, columbine shooting, home depot, vaginal dryness, black book, weather map, marilyn monroe, jack cafferty, mary winkler, wall-street journal, hot flashes, illegal immigration, andrea yates, 7 wonders of the world, big lots, mikhaïl gorbachev, freedom tower, buck owens, kanye west, tom delay, coretta scott king, andrew luster, end times, home improvement, operation swarmer, scott peterson, the cw network, weather channel, aaron brown, soul train awards, craig s list, gas mileage

Figure A.6: Phrase queries used for NYT dataset

List of Figures

| | | |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 2.1 | Components of an inverted index | 15 |
| 2.2 | Example Hidden Markov Model | 27 |
| 2.3 | New York Times article annotated using TARSQI | 31 |
| 2.4 | LHAM's rolling merge illustrated | 37 |
| 2.5 | URL http://www.mpi-sb.mpg.de as of March 5, 1997 archived by the Internet Archive (http://www.archive.org) | 41 |
| 3.1 | Time-Travel Inverted index (TTIX) example instance | 51 |
| 3.2 | Physical storage of posting list <code>cat:[3, 6)</code> from Figure 3.1 | 52 |
| 3.3 | Temporal coalescing for scalar payloads illustrated | 61 |
| 3.4 | Document versions d^{t_1} , d^{t_2} , and d^{t_3} containing words a , b , and c | 65 |
| 3.5 | Situation impossible according Lemma 3.5 | 72 |
| 3.6 | Partitioning illustrated | 75 |
| 3.7 | Situation impossible according to Lemma 3.6 | 81 |
| 3.8 | FLUXCAPACITOR's System Architecture | 89 |
| 3.9 | FLUXCAPACITOR's Web-based GUI showing query results for the query <code>iraq war</code> and different query time-points | 91 |
| 3.10 | Excerpts from query workloads | 98 |
| 3.11 | Result accuracy at cut-off level $k = 5$ on WIKI and UKGOV | 107 |
| 3.12 | Result accuracy at cut-off level $k = 10$ on WIKI and UKGOV | 108 |
| 3.13 | Result accuracy at cut-off level $k = 25$ on WIKI and UKGOV | 109 |
| 3.14 | Result accuracy at cut-off level $k = 100$ on WIKI and UKGOV | 110 |
| 3.15 | Index size and expected processing cost for Boolean queries on WIKI, UKGOV, and NYT-30 | 124 |
| 3.16 | Index size and expected processing cost for keyword queries on WIKI, UKGOV, and NYT-30 | 125 |
| 3.17 | Index size and expected processing cost for phrase queries on WIKI, UKGOV, and NYT-30 | 126 |

| | | |
|-----|----------------------------------------------------------------------------------------------------------------------------------------|-----|
| 4.1 | iPod@2005 and Walkman@1990 with frequently co-occurring terms | 135 |
| 5.1 | Documents from The New York Times relevant to the query <i>fifa world cup 1990s</i> likely to be missed by existing retrieval models . | 154 |
| 5.2 | Three requirements for a generative model | 161 |
| 5.3 | Amazon Mechanical Turk HIT to collect queries by letting users fill in an entity that fits a given temporal expression | 169 |
| 5.4 | Amazon Mechanical Turk HIT to collect queries by letting users fill in a temporal expression that fits a given entity | 170 |
| 5.5 | Queries categorized according to topic and temporal granularity | 171 |
| 5.6 | Amazon Mechanical Turk HIT to collect relevance assessments . | 172 |
| A.1 | Boolean/Keyword queries used for WIKI dataset | 201 |
| A.2 | Boolean/Keyword queries used for UKGOV dataset | 202 |
| A.3 | Boolean/Keyword queries used for NYT dataset | 203 |
| A.4 | Phrase queries used for WIKI dataset | 204 |
| A.5 | Phrase queries used for UKGOV dataset | 205 |
| A.6 | Phrase queries used for NYT dataset | 206 |

List of Tables

| | | |
|------|--------------------------------------------------------------------------------------------------------------|-----|
| 2.1 | Summary of notation | 24 |
| 3.1 | Websites contained in the UKGOV dataset | 95 |
| 3.2 | Dataset statistics (with mean μ and standard deviation σ) | 96 |
| 3.3 | Relevant domain suffixes per dataset | 97 |
| 3.4 | Impact of temporal coalescing on index size for Boolean payloads on WIKI and UKGOV | 100 |
| 3.5 | Impact of partitioning strategies on index size for Boolean payloads on WIKI, UKGOV, and NYT-30 | 100 |
| 3.6 | Impact of temporal coalescing on index size for scalar payloads on WIKI and UKGOV | 102 |
| 3.7 | Impact of partitioning strategies on index size for scalar payloads on WIKI, UKGOV, and NYT-30 | 102 |
| 3.8 | Impact of temporal coalescing on index size for positional payloads on WIKI and UKGOV | 105 |
| 3.9 | Impact of partitioning strategies on index size for positional payloads on WIKI, UKGOV, and NYT-30 | 105 |
| 3.10 | Impact of temporal coalescing on the processing performance of Boolean queries on WIKI and UKGOV | 113 |
| 3.11 | Impact of partitioning strategies on the processing performance of Boolean queries on WIKI | 114 |
| 3.12 | Impact of partitioning strategies on the processing performance of Boolean queries on UKGOV | 115 |
| 3.13 | Impact of partitioning strategies on the processing performance of Boolean queries on NYT-30 | 115 |
| 3.14 | Impact of temporal coalescing on the processing performance of keyword queries on WIKI and UKGOV | 117 |
| 3.15 | Impact of partitioning strategies on the processing performance of keyword queries on WIKI | 118 |

| | | |
|------|-----------------------------------------------------------------------------------------------------------|-----|
| 3.16 | Impact of partitioning strategies on the processing performance of keyword queries on UKGOV | 119 |
| 3.17 | Impact of partitioning strategies on the processing performance of keyword queries on NYT-30 | 119 |
| 3.18 | Impact of temporal coalescing on the processing performance of phrase queries on WIKI and UKGOV | 121 |
| 3.19 | Impact of partitioning strategies on the processing performance of phrase queries on WIKI | 121 |
| 3.20 | Impact of partitioning strategies on the processing performance of phrase queries on UKGOV | 122 |
| 3.21 | Impact of partitioning strategies on the processing performance of phrase queries on NYT-30 | 122 |
| 4.1 | Benchmark terms used in our experimental evaluation | 145 |
| 4.2 | Benchmark queries used in our experimental evaluation | 146 |
| 4.3 | Terms reported as most across-time semantically similar | 147 |
| 4.4 | Top-3 across-time query reformulation results | 148 |
| 5.1 | Dataset statistics (with mean μ and standard deviation σ) | 168 |
| 5.2 | Retrieval effectiveness overall on NYT | 174 |
| 5.3 | Retrieval effectiveness overall on WIKI | 174 |
| 5.4 | Retrieval effectiveness by topic on NYT | 175 |
| 5.5 | Retrieval effectiveness by topic on WIKI | 175 |
| 5.6 | Retrieval effectiveness by temporal granularity on NYT | 176 |
| 5.7 | Retrieval effectiveness by temporal granularity on WIKI | 176 |

List of Algorithms

| | | |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 1 | Viterbi algorithm | 29 |
| 2 | Determining an optimal sequence \mathcal{L}_v of time intervals whose posting lists are merged to retrieve all postings relevant to query term v and query time-interval $[t_b, t_e]$ | 57 |
| 3 | Temporal coalescing for scalar payloads | 63 |
| 4 | Temporal coalescing for positional payloads (optimal) | 69 |
| 5 | Temporal coalescing for positional payloads (approximate) | 71 |
| 6 | Performance-guarantee partitioning (optimal) | 78 |
| 7 | Performance-guarantee partitioning (approximate) | 80 |
| 8 | Space-bound partitioning (optimal) | 84 |
| 9 | Space-bound partitioning (approximate) | 86 |
| 10 | Computing the best-k state sequences | 141 |

Index

- FLUXCAPACITOR, 89
- 7-Bit Encoding, 16, 94
- A* Search, 140
- Across-Time Semantic Similarity, 135
- Amazon Mechanical Turk,
 see Crowdsourcing
- AOL Query Logs, 96
- B-Tree, 14, 34
- Back to the Future Trilogy, 89
- Boolean Query, 8
- Boolean Retrieval, 8

- Chronon, 158
- Cornell Web Library, 46
- Cosine Similarity, 9
- Cross-Language Information Retrieval,
 133
- Crowdsourcing, 157
 - Amazon Mechanical Turk, 157, 168,
 170
- Data Exploration, 180
- Data Mining, 180
- DF, *see* TF·IDF
- Dijkstra's Algorithm, 54
- Document-at-a-Time, *see* Query Processing
- Dynamic Programming, 27, 68, 78,
 85

- Elias- δ Encoding, 16
- Elias- γ Encoding, 16
- European Archive, 40
 - UKGOV, 95
- Fleiss' κ , 22, 170
- Gap Encoding, 17, 94
- Google Timeline View, 157
- GUTime, 30

- Hidden Markov Model, 26
- HMM, *see* Hidden Markov Model

- IDF, *see* TF·IDF
- IIPC, 40
- Internet Archive, 40
- Inverted Index, 14, 44
 - Lexicon, 14
 - Payloads, 15
 - Posting Lists, 14
 - Posting-List Order, 15
 - Postings, 14

- Java, 94, 145, 166

- Kendall's τ , 21
- Keyword Query, 8–11
- Kullback-Leiber Divergence, 26

- Language Model, 11, 159
 - for Temporal Expressions, 159
 - Smoothing, 160
 - Uncertainty-Aware, 163
 - Uncertainty-Ignorant, 162
 - Smoothing, 11
- LHAM, *see* Temporal Databases
- Link Analysis, 13
- MVBT, *see* Temporal Databases
- nDCG, *see* Normalized Discounted Cumulative Gain
- New York Times Annotated Corpus, 95, 144, 167
- Normalized Discounted Cumulative Gain, 20
- NRA, *see* Query Processing
- Okapi BM25, 10, 49, 180
- Oracle, 145, 166
- PageRank, 13
- Peer-to-Peer, 181
- Phrase Query, 12
- Precision, 20
- Prefix Sums, 88
- Proximity, 12
- Query Paraphrasing, 132
- Query Expansion, 132
- Query Processing
 - Document-at-a-Time, 18
 - Early-Terminating Methods, 18
 - NRA, 18
 - Term-at-a-Time, 17
- Query Refinement, 132
- Query Reformulation, 137
- Query-Likelihood Approach, 159
- Ramaswamy's Approach, *see* Temporal Databases
- Recall, 20
- Relative Recall, 21
- Semantic Similarity, 24
- Simulated Annealing, 86
- Strong Contextual Hypothesis, 24
- TARSQI, 30
- Temporal Coalescing, 58
 - for Boolean Payloads, 59
 - for Positional Payloads, 64
 - for Scalar Payloads, 60
 - in Temporal Databases, 39
- Temporal Databases, 32
 - LHAM, 36
 - Multi-Version B-Tree, 34, 44, 47
 - Ramaswamy's Approach, 38
 - Temporal Coalescing, 39
 - Transaction Time, 32
 - TSB-Tree, 34, 44, 47
 - Valid Time, 33
- Temporal Expression, 28, 158
 - Explicit, 28
 - Implicit, 28
 - Relative, 30
- Temporal Information Need, 153
- Temporal Partitioning, 73
 - Performance Guarantee, 76
 - Performance-Optimal, 75
 - Space Bound, 83
 - Space-Optimal, 76
- Term-at-a-Time, *see* Query Processing
- TF, *see* TF-IDF
- TF-IDF, 9

-
- Time Frames, 157
 - Time-Dependent Collection Statistics, 88
 - Time-Travel Inverted Index, 50, 92
 - Posting Lists, 50
 - Posting-List Order, 53
 - Postings, 50
 - Query Processing, 53
 - Temporal Partitioning, 51
 - Time-Interval Queries, 54
 - Time-Point Queries, 53
 - Time-Travel Text Search, 43
 - TimeML, 30
 - TimeSearch, 157
 - TimexTag, 30
 - Transaction Time, *see* Temporal Databases
 - TSB-Tree, *see* Temporal Databases
 - TTIX, *see* Time-Travel Inverted Index
 - Unary Encoding, 16
 - UNIX Epoch, 48, 158
 - Valid Time, *see* Temporal Databases
 - Vector Space Model, 8
 - Viterbi Algorithm, 27, 141
 - Wikipedia
 - Revision History 2001–2005, 95
 - Snapshot as of July 2009, 167
 - Zoetrope, 46