Summarizing Temporal Data with Approximate Temporal Aggregation and Ranking

Juozas Gordevičius

Summarizing Temporal Data with Approximate Temporal Aggregation and Ranking

TESI PER IL DOTTORATO DI RICERCA IN INFORMATICA

DOKTORARBEIT IN INFORMATIK

PHD THESIS IN COMPUTER SCIENCE

Juozas Gordevičius

Relatore - *Doktorvater* - Faculty Advisor: Prof. Johann Gamper
Correlatore - *Zweitbetreuer* - Faculty Co-Advisor: Prof. Michael H. Böhlen

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

Fakultät für Informatik | **Facoltà di Scienze e Tecnologie informatiche** | **Faculty of Computer Science**

Facoltà di Scienze e Tecnologie Informatiche
Libera Università di Bolzano
Piazza Domenicani 3
39100 Bolzano
Italia

Fakultät für Informatik
Freie Universität Bozen
3 Domenikanerplatz
39100 Bozen
Italien

Faculty of Computer Science
Free University of Bozen-Bolzano
Piazza Domenicani 3
39100 Bozen-Bolzano
Italy

tel: +39 0471 016 000
fax: +39 0471 016 009
e-mail: cs-secretariat@unibz.it
homepage: http://www.unibz.it/inf

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

This thesis and related publications have evolved piece-by-piece in various places around the world. The people I met and made friends with during my travels contributed in many various ways to the formation of my ideas and the creation of these texts. I would like to express my most sincere gratitude to all these people.

In particular I would like to thank my supervisor, Professor Johann Gamper, for his infinite patience, guidance, discussions and funding throughout my PhD studies. I am in debt to the whole Faculty of Computer Science at the Free University of Bozen-Bolzano; in particular Professor Michael Böhlen who found the time to review all my papers and Mouna Kacimi who was always ready to wade into a discussion.

Part of this research work was conducted at the offices of Thoora Inc. in Toronto, Canada. I am particularly grateful to my colleagues there, especially Chul Lee, Periklis Andritsos, Francisco Estrada, Darius Braziunas and Ksenia Shubina. We spent a lot of time discussing ranking issues, eating sushi and chinese dumplings, as well as exploring the national parks of Canada. Without them the time spent in Toronto would have been dull and tedious.

I would like express my gratitude to the sailors of Toronto Sailing and Canoe Club, Ruth Watkins and Judy Vineyard, as well as their families and friends I had an opportunity to meet. I hope Ruth's relentless efforts to improve my english have not been futile.

Last, but not least, I am grateful to my beloved family. Without their love and support I would have not been able to finish this work.

Juozas Gordevičius

November, 2010.

Abstract

In this thesis we study issues related to large scale analysis of historical data. We observe that summarizing such data and discovering the most important aspects of it are the key issues to address. Temporal aggregation is used to summarize time varying information. However, the two most prominent operators in use today do not meet the demands of large scale applications. Instant temporal aggregation tends to produce more output data than the input, which is contrary to the very idea of aggregation. Span temporal aggregation takes non-data adaptive measures to control the size of the output. Therefore, we propose a new aggregation concept, parsimonious temporal aggregation (PTA), that overcomes the limitations of existing approaches. PTA takes a data adaptive approximation approach, allowing the user to control the tradeoff between the size of the aggregation result and the error induced by the consequent approximation.

We propose a dynamic programming algorithm for the offline evaluation of PTA queries that benefits from temporal gaps present in the data and aggregation groups specified in the query. The computation complexity of the algorithm, although quadratic in the worst case, is reduced to linear for real-world situations. For the online evaluation of PTA queries we propose a time and space efficient greedy merging algorithm that reduces n input tuples to c in $O(n \log(c+\beta))$ time using $O(c+\beta)$ space, where β is typically very small. The efficiency of the greedy approach comes at a slight loss of approximation precision that we prove to be upper bounded by $O(\log n)$. An empirical evaluation using synthetic and real world data shows that PTA considerably reduces the aggregation result introducing only small errors. The greedy solution is scalable for large datasets and induces less approximation error than other commonly used approximation techniques.

The second issue addressed in this thesis concerns a framework for mining the aggregation groups from data, and for ranking them. As an application scenario we we focus on the problem of ranking news stories

within their historical context by exploiting their content similarity. We observe that news stories evolve and thus have to be ranked in a time and query dependent manner. We do this in two steps. First, the mining step discovers metastories, which constitute meaningful groups of similar stories that occur at arbitrary points in time. Second, the ranking step uses well known measures of content similarity to construct implicit links among all metastories, and uses them to rank those metastories that overlap the time interval provided in a user query. We use real data from conventional and social media sources (weblogs) to study the impact of different meta-aggregation techniques and similarity measures in the final ranking. We evaluate the framework using both objective and subjective criteria, and discuss the selection of clustering method and similarity measure that will lead to the best ranking results.

The combined application of the PTA operator and the ranking framework allows us to store and analyze aggregated data over long periods of time. The data aggregated with PTA will take less space yet preserve the most important information. The ranking framework will allow the selection of the most important aspects of the data depending on the user-specified historical context.

CHAPTER 1

Introduction

1.1 Motivation

Data analysis is at the core of many business, scientific and administrative processes. For example, data analysis allows companies to discover new, previously unknown, information about the behavior and needs of their clients and adapt the business accordingly. In scientific research, data collected from sensors may be used to prove or formulate novel hypotheses. A significant amount of computer science and database research effort lead to techniques and industry-standard tools that facilitate data collection and analysis. Relational databases are now used for data storage and SQL is the most widely used query language used in database retrieval.

In almost every transactional database a timestamp or a validity time interval is assigned to data records. Employment databases store contract start and duration as a temporal validity interval. Observational data bears the timestamp indicating the time the measurements took place. Medical databases record patient histories and their visits to doctors. Stock market databases track the commodity prices that change over time. News aggregation engines store the time an article was published, as well as the time it was retrieved by the web crawler.

The temporal dimension captured in the data enables the users to analyze and possibly predict the way the system under consideration changes over time. It also allows one to identify trends and repeating patterns. Temporal aggregation aims at summarizing the data and enabling the analysis. It differs from traditional aggregation as aggregate values are computed over a set of time intervals. Traditional database tools, however, do not allow efficient temporal aggregation [34]. Although several temporal aggregation techniques have been proposed to address the issue, the aggregated result they provide is overly detailed. Today, when the amount of archived data is enormous, temporal aggregation should efficiently provide a compact overview of all the data available, putting the emphasis on the most significant changes. Moreover, it should use the underlying data to select the aggregation intervals and aggregation groups instead of leaving this task to the user, as is the case with current techniques.

In fact, the need to monitor the evolution of a system under consideration has recently captured the interest of the general public, media, industry and scientists alike. In various fields, from climate to economics, finding out what has changed will be fundamental in understanding why it happened. For example, today many people feel that human activity across the planet has lead to global climate change. Whether or not this is true, what caused the change and what the future state will be, are subject to much scientific and public debate. However, it is clear that adequate large scale monitoring and analysis tools, as well as international policies, must be in place for a consensus on the cause and future state to be reached [25].

Accordingly, recent years have seen increased interest in historical data

management and temporal databases among scientific communities and industry . As an example, consider a state-of-the-art content aggregation platform that was recently unveiled by Thoora[1]. The Thoora platform solves the problem of how to cluster together news articles, blogs and tweets all relating to a given story. Thus, existing technology for news-related content aggregation allows users to have easy access to current stories, and to benefit from the wide range of opinions and comments provided by social media. However, the larger-scale problem of organizing, grouping and searching historical archives of such and similar content remains largely unsolved due to the enormous amount of information that has to be handled and the evolutionary dynamics of the temporal data itself.

A recent article by the BBC[2] debates the most important events of the last decade. In this article the authors ask the readers to list the most important events of the last decade and rank them. Unfortunately, while the events of 911, the launch of the iPhone, etc. received their mention, the launch of YouTube, or the sequencing of the human genome had been forgotten and not included on the list. Moreover, as can be seen in Fig. 1.1, some events pertain to distinctly different news categories, therefore, it is hard to judge which one should be ranked higher. In summary, the article shows that public memory is not a reliable approach to rank historical events. The authors conclude that "any attempt to list the major events of a decade is almost bound to disappoint. It will miss things that some will consider vital." We feel that the above conclusion is premature. And we argue that in order to answer such and similar questions we must collect historical data and learn how to analyze it in a principled and automated way.

1.2 Problem Statement

Today data warehouses accommodate data that streams in vast amounts from a plethora of heterogenous sources. For example, the above mentioned news aggregation system combines different sources of user opinion, i.e. blogs and tweets, with articles from conventional media into one database. In most cases the incoming data records are stored together with timestamps or temporal validity intervals. This temporal dimension is crucial as it enables the analysis of the collected data with a temporal perspective. Temporal aggregation is used to provide the user with a summary of the data. It allows one to identify trends, and search for patterns and other time related developments that would otherwise remain invisible.

Various forms of temporal aggregation have been studied in the past. Most importantly, instant temporal aggregation (ITA) [12, 34, 43, 55, 42, 60]

[1] www.thoora.com

[2] BBC, 14 December 2009, http://news.bbc.co.uk/2/hi/8409040.stm

Figure 1.1: Is it really possible to sum up a decade?

and span temporal aggregation (STA) [12, 34]. The value of ITA at time instant t is computed from the set of tuples that hold at t. Consecutive time points with identical aggregate values are then coalesced. Thus, ITA works at the smallest time granularity. Its main drawback is that the result size depends on the argument relation, and may grow to be twice as large as the latter [12]. This behavior is in conflict with the very idea of aggregation, which is to provide a summary of the data.

Example 1.1. As an example consider relation **proj** in Figure 1.2(a) that records information about project contracts, i.e. the name of an employee, the project he/she works for, the monthly salary, and the time period (T) (in months) during which the contract is effective. The tuple r_1 states that John received a salary of 800 for every of the four months when he worked for project A. The validity intervals are illustrated as horizontal lines. Figure 1.2(b) shows the result of an ITA query *"What is the average monthly salary for each project?"* containing one grouping and one aggregation attribute. Its size exceeds the size of the input relation, though some adjacent tuples have quite similar aggregate values.

Span temporal aggregation (STA) allows one to control the result size by permitting an application to specify the time intervals in which to report a result tuple, e.g., for each year from 2000 to 2005. For each of these intervals a result tuple is produced by aggregating over all argument tuples that overlap that interval. STA might not always provide good summaries of the data, since the intervals are specified *a priori* without considering the distribution of the data.

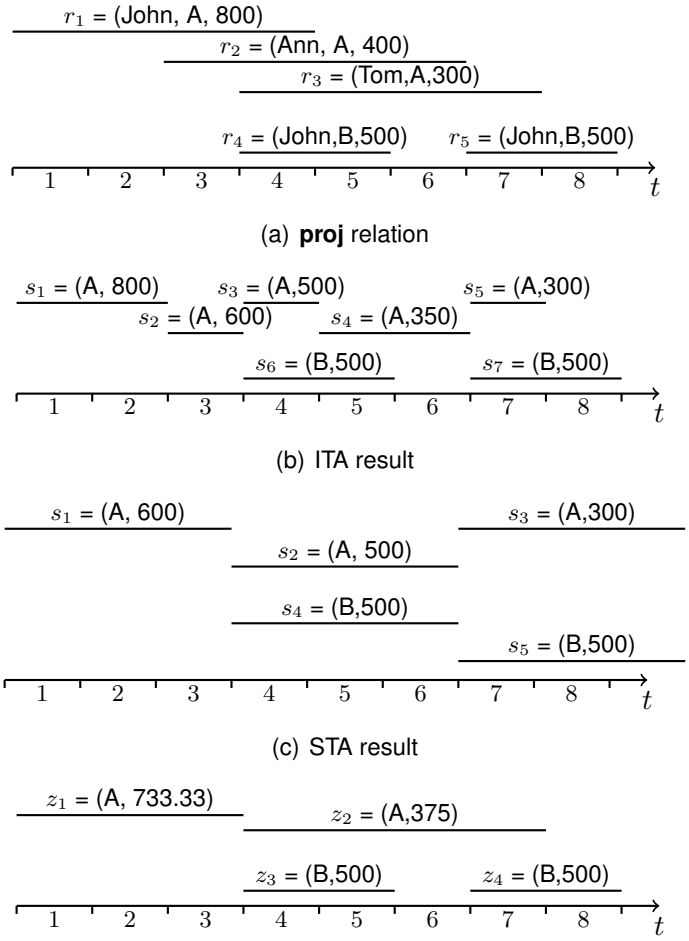


Figure 1.2: Temporal aggregation over the **proj** relation

Example 1.2. The result of STA query *"What is the average monthly salary for each project at each trimester?"* is shown in Figure 1.2(c). Evaluation of the query involves constructing one result tuple per three month period and aggregation group. Since the aggregation intervals are imposed over the data, the approach is likely to miss important changes that do not dominate the interval.

Both, ITA and STA, do not meet the demands of modern applications. They fail to provide good summaries of vast temporal datasets. Their drawbacks hinder visualization, similarity search and, consequently, other temporal data mining tasks, e.g., clustering. Therefore, we need a novel approach to temporal aggregation that would provide compact summaries of the data and introduce data adaptive approximations, i.e. such that are dictated by the data and not imposed over it.

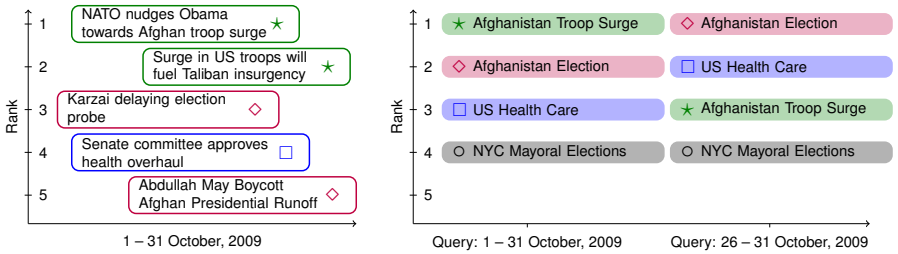


Figure 1.3: Effect of the query time interval on the ranking result

The second problem is that the grouping used in the temporal aggregation process is unlikely to be known *a priori*. In fact, contrary to traditional aggregation, the aggregation groups evolve, i.e. appear, merge, and disappear as the time goes. We observe that they form very complex structures that have to be discovered from the underlying data and not imposed by the user. Consider the previously mentioned news aggregation scenario. The users may wish to follow particular stories as they develop through weeks or even months. Each story, therefore, forms an aggregation group. Within every group we compute a time varying aggregate representation of all the articles, blogs and tweets that belong to it. Typically, however, already at the data collection step, such stories will be broken up into several separate events and form separate aggregation groups. This may occur because important stories have multiple ramifications and branch into components that are different enough not to be clustered together. It can also occur because of the temporal separation between developments that are part of the same story. While keyword search may retrieve many of these in response to a query, the true importance of the whole story cannot be evaluated if its components have not been linked. Thus existing news aggregators cannot accurately rank the search results.

Example 1.3. In Figure 1.3 we show an example of the news story aggregation and ranking process. Individual story elements, shown in Figure 1.3(a), were gathered during the month of October 2009. The figure shows only five out of the 700 news-related clusters that were used. Even so, notice that the individual clusters can be grouped into metastories corresponding to important events occurring during the month[3].

In summary, modern applications require a temporal aggregation solution that efficiently aggregates vast datasets allowing the user control over the size of the aggregation result or the overall approximation error. In addition, automated techniques for discovering aggregation groups to be used

[3]Without loss of valuable information, we use the most descriptive titles for the stories and metastories in Figure 1.3

for the aggregation are necessary. Since the groups evolve and change over time we need a method to disseminate the most important ones based on the time interval of interest.

1.3 Proposed Solution and Contributions

The solution to the above problem that we propose in this thesis is comprised of two components. First, we define a novel temporal aggregation operator, termed parsimonious temporal aggregation (PTA). It approximates the aggregation result in a data adaptive fashion allowing one to discard small fluctuations of the aggregate result that are of no interest to the application and, instead, report (longer) time periods without significant changes. Second, we introduce a temporal ranking framework allowing for the mining and ranking of the aggregation groups from the input data. The combined application of the PTA operator and the temporal ranking framework presents the user with the information that reflects the most significant and interesting changes in aggregate values over time. Such information will greatly simplify further data mining tasks, such as visualization, classification, clustering and trend analysis. In the following two subsections we provide a brief overview of the two components.

1.3.1 Parsimonious Temporal Aggregation

Parsimonious temporal aggregation is based on the following observation: an application is typically not interested in small fluctuations of the aggregation result, but in (longer) time periods without significant changes. PTA comprises two main steps: (1) it computes the ITA result over the input relation and, (2) it reduces this intermediate result to a user-specified size c. The reduction is done by merging only time-wise adjacent tuples that belong to the same aggregation group and keeping the induced total error minimal. The compressed ITA result is returned as the final PTA result. The PTA operator allows for the control of the size of the output relation, thus overcoming the main limitation of ITA. On the other hand, it induces the best time intervals from data thus overcoming the main limitation of STA.

Example 1.4. Figure 1.2(d) shows the PTA result for the query *"What is the average monthly salary for each project represented in 4 tuples?"*. Three merging steps were taken to obtain the result. In contrast to the ITA result, the PTA tuples reveal significant changes in the aggregation values.

In this thesis we formally define the PTA operator for two types of user input, namely the size- and error-bound. Size-bounded PTA reduces the ITA relation to a user specified size minimizing the total error, whereas error-bounded PTA minimizes the result size under a user specified total

error threshold. We investigate various strategies for evaluation of PTA queries and propose algorithms for two fundamental scenarios – offline and online query evaluation. In an offline evaluation scenario the user is interested in the most precise PTA evaluation and is not overly limited by memory and computing power. In an online scenario, on the other hand, the user is interested in obtaining the aggregation result as quickly as possible and is willing to sacrifice some precision in exchange for speed and and the amount of memory required. We propose dynamic programming and greedy algorithms tailored for both offline and online scenarios respectively. Most importantly, we show how temporal grouping in the query and temporal gaps in the data allow significant speedup of these algorithms. Last, but not least, we prove that the additional approximation error introduced by the greedy strategy is upper-bounded and likely to be very small.

The experimental evaluation reveals that real-world temporal data tends to have a great deal of redundant information. In some cases we are able to reduce the relation to 5% of its size maintaining the aggregation error close to 0. The greedy algorithms show exceptional approximation quality outperforming its most similar competitor, approximate temporal coalescing (ATC) [8] and other state-of-the art approximation methods used in time series data analysis.

1.3.2 Temporal Ranking Framework

We introduce temporal ranking framework which is crucial in dynamic environments where the aggregation groups used for temporal aggregation are not known in advance. Typically in such environments the groups form very complex structures that evolve in time and the user is only interested in the most important ones. Their ranking, however, depends on the time interval specified in the aggregation query. For example, the most important events of this week are different from the ones of the whole year. Therefore, the framework consists of online and offline components. The offline component discovers all the aggregation groups. The online ranking component will rank only those groups that fall into the time frame of the particular user query.

We study the temporal ranking problem in the context of the news aggregation engine Thoora, cited previously. We propose a novel framework that allows the user to follow large-scale events as they develop through time. The problem is very challenging and has offline and online components. Offline, given (a) a set of stories with their valid time intervals and, (b), their content-based similarity, we find a way to cluster similar ones into metastories. Online, we determine which metastories are the most relevant in the context of user specified time frame.

The clustering task is challenging because, as noted before, the different components of a story have already been deemed to be distinct enough

by the aggregator to be put into different story clusters. Ranking is difficult because the relevance of events is not static – it depends on the specific period a user is interested in, and how concurrent events during that timespan evolve in terms of social impact. In other words, a story that captured the attention of the whole world last month will not necessarily be considered as one of the most important stories of the decade.

Example 1.5. In Figure 1.3 we show how metastory aggregation is performed over the set of all existing story components, and is independent of any interval a user may be interested in. Once metastories have been constructed, the ranking step determines their relative order with regard to a time interval specified by the user. Figure 1.3(b) shows two sample rankings for different query intervals in 2009: Oct. 1-31, and Oct. 26-31. Notice that the relative ranking of metastories changes. In particular, the "Afghanistan Troop Surge" metastory is the most important overall story of October 2009, but when only the last week of the month is considered, its relevance is lower compared to other metastories that were more active during this specific period. Such relationships can only be discovered through meta-aggregation and time-dependent ranking. Given the same set of metastories, a content-based search would produce the same ranking independent of the period in which the user was interested. In summary, the time-dependent nature of the ranking process, coupled with the more comprehensive scope of metastories, yields a more informative and more representative ranking of news events.

In this thesis we determine which clustering method and similarity measure produce the best metastories. We evaluate several existing clustering methods [21, 56, 49] that have proven successful in different domains. At the same time we explore the issue of measuring similarity between separate story components. Along the way we look at the statistical behavior of similarity measures on typical data available to information retrieval and content aggregation platforms.

Next we explore the ranking problem proposing that ranking can be fully independent of clustering if proper formulation is used. We use a graph-based formulation and where the rank of a node is high if other highly important nodes link to it. Since blog and news sites rarely link to each other or to their sources of information, we propose to use their similarity to create the graph edges. Thus the weights of the edges are set to the similarity among the nodes. Ranking is then evaluated using the PageRank algorithm and exploiting a known fact that for symmetric similarity matrices the PageRank computation is trivial [16]. Hence, ranking is reduced to the problem of computing a good similarity matrix capturing the strength of the relationship between different story components.

Finally, we evaluate the performance of different similarity measures within this framework to determine which measure yields the best ranking

compared to the actual relevance of each metastory. In our setting the actual relevance corresponds to the total social impact the story has experienced. To measure it we exploit the fact that Thoora has access to social media, and that it can relate specific blog and tweet information to particular events.

1.4 Related Publications

The thesis covers the research work conducted at the Free University of Bozen–Bolzano during 2007-2010 under the supervision of Professor Johann Gamper and in collaboration with Professor Michael Böhlen. The ranking work was conceived in collaboration with Thoora Inc. (Toronto, Canada) under the supervision of Periklis Andritsos.

Most of the contributions presented in this thesis have been published (or are in the process of publication) as the following papers:

- J. Gordevicius, J. Gamper, and M. Böhlen. A Greedy Approach Towards Parsimonious Temporal Aggregation. In Proc. of the 15th International Symposium on Temporal Representation and Reasoning (TIME-08), pages 88 – 92, June 2008.
- J. Gordevicius, J. Gamper, and M. H. Böhlen. Parsimonious Temporal Aggregation. In Proc. of the International Conference on Extending Database Technology (EDBT), pages 1006 – 1017, 2009.
- J. Gordevicius, P. Andritsos, F. Estrada, Hyun C. Lee, J. Gamper. Ranking of Evolving Stories Through Meta-Aggregation. In Proc. of 19th International Conference on Information and Knowledge Management (CIKM), pages 1909–1912, 2010.
- J. Gordevicius, J. Gamper, and M. H. Böhlen. Parsimonious Temporal Aggregation. Under review at VLDB Journal.

1.5 Outline

After this introduction we continue with the formal discussion of the novel temporal aggregation operator and temporal ranking framework. Chapter 2, presents an overview of the state of various research fields related to our problem. Once we are familiar with the related works, we formally define the parsimonious temporal aggregation operator in Chapter 3 and corresponding evaluation algorithm. For situations when PTA queries have to be evaluated very quickly, and some precision can be sacrificed, we propose a novel greedy evaluation strategy in Chapter 4. Next, we turn to mining and ranking of aggregation groups for temporal aggregation. We propose our

temporal ranking framework in Chapter 5. Finally, we list the contributions made in this thesis and conclude the work in Chapter 6.

CHAPTER 2

State of the Art

In this chapter we present the research works that are related to the topic of this thesis or address similar problems. We delve into the recent developments in the fields of temporal aggregation, histogram construction, time series approximation, topic detection and tracking and, finally, ranking approaches used in information retrieval.

2.1 Overview

The temporal aggregation, approximation and ranking problems addressed in this thesis are related to similar problems in various other fields. The solutions we propose here leverage from other approaches or can be compared to them for performance and efficiency. Therefore, in this chapter we review these relevant solutions in the fields of temporal aggregation, histogram construction, time series approximation, topic detection and tracking and ranking approaches used in information retrieval.

A plethora of temporal aggregation operators have been proposed in the temporal database research field. We present and discuss the different aspects, advantages, and drawbacks of the three most prominent ones, namely instant, span and moving-window aggregation operators. The parsimonious temporal aggregation (PTA) operator, that we will introduce in Chapter 3, overcomes the limitations of these approaches by effectively approximating the result of instant temporal aggregation. The algorithm we propose for evaluation of PTA queries is based on achievements in the histogram construction research area. The dynamic programming strategy proposed by Jagadish et al. [27] for optimal histogram construction inspired also our approach.

Later in this thesis we introduce a greedy strategy for the evaluation of PTA queries tailored for online environments where a quick answer is of the utmost importance. We compare the greedy approach to other linear approximation methods proposed in the field of time series approximation. The field has seen very active development in the recent years and various approximation techniques ranging from discrete wavelet transform to symbolic approximate representation have been proposed. Most of these techniques can be applied to evaluate a certain subclass of PTA queries. Therefore, in this section we review these techniques.

The temporal ranking framework that we propose has its roots in various disciplines. First, clustering of textual documents is often used in topic detection and tracking (TDT) to discover documents that relate to the same topic and to identify new topics. We will discuss the recent work in this field and explain how our approach differs from the general TDT strategy. Finally, ranking of textual information retrieved by search engines is at the core of information retrieval. Recent years have seen increased interest in the problem with such algorithms as PageRank and HITS changing the way we understand the principles of ranking.

2.2 Temporal Databases and Aggregation

Relations in temporal databases reflect the time varying nature of the data. Models of such databases have been studied extensively and are well doc-

umented in a book by Tansel et al. [52]. In 1998 Jensen et al. proposed terminology to be used to name temporal database concepts [28]. For example, they distinguish between *transaction time*, that defines when the tuple was written to disk, and *valid time*, that allows one to determine when the data specified in the tuple was valid. In this work, most importantly when defining the parsimonious temporal aggregation in Chapter 3, we use the temporal database model and terminology that adhere to the above works.

Various forms of temporal aggregation have been studied in the past. The most prominent approaches are the instant, span and moving-window temporal aggregation. They differ mainly in how the timeline is partitioned.

Instant temporal aggregation (ITA) [34, 43, 42] operates at the smallest time granularity. The timeline is partitioned into time instants, and for each time instant, t, and aggregation group, g, the aggregate functions are evaluated over all tuples that hold at t and belong to g. Then, identical aggregate results for consecutive time instants are coalesced into tuples over maximal time intervals. While ITA reports the most detailed result, its main drawback is that the result size that is typically larger than the argument relation, and can be up to twice the size of it.

Moving-window temporal aggregation (MWTA) [45], later termed cumulative temporal aggregation [50, 60], extends ITA. For each time instant, t, it computes the aggregate values over all tuples that hold in a window "around" t. Just like ITA, it is prone to returning large result relations.

Span temporal aggregation (STA) [50] allows for the control of the result size by partitioning the timeline into predefined intervals. For each such interval and aggregation group, a single result tuple is produced by evaluating the aggregate functions over all argument tuples that overlap that interval and belong to the group. However, the timestamps of the result tuples are specified by the application and are independent of the argument data. Most approaches consider only regular time spans expressed in terms of granularities, e.g., years or months.

Most research works address the efficient evaluation of ITA queries [34, 43], and cumulative aggregates [60]. Temporal aggregates with additional range predicates [62] receive less attention according to [12]. The first approach to temporal aggregate computation was introduced by Tuma [55]. The computation requires two scans of the argument relation. Kline and Snodgrass [34] later show that temporal aggregation using traditional SQL is not efficient, thereby motivating the development of a separate algorithmic approach. They propose a solution that requires only one scan of the argument relation and builds a main-memory based aggregation tree structure. Since the tree is not balanced, the worst case complexity is $O(n^2)$ for n tuples.

Moon et al. [43] propose the use of a balanced aggregation tree. The solution works only for $sum, count$, and avg functions, thus for min and max they proposed a merge-sort like algorithm. Both algorithms run in

$O(n \log n)$ time thus improving over the previous approaches. The authors also propose a scalable solution for large relations that do not fit into main memory. They assign tuples to buckets according to the partitioning of the timeline and perform aggregation on each bucket separately. The approach takes into account long-lived tuples, however, it requires three scans of the whole argument relation.

Yang and Widom [60] advanced the evaluation of temporal aggregates even further. They proposed a method that supports disk-based incremental computation and maintenance of instant and span temporal aggregation results. They introduce a SB-Tree structure that incorporates features from segment trees and B-Trees. It constructs a hierarchy of intervals maintaining within each interval a partially computed aggregate. The tree does not support selection predicates, that is, aggregate queries may only be applied to the entire base relation. The drawback is addressed by Zhang et al. [62] with a multi-version SB-Tree.

The later works [42, 12] seek to unify different aggregation operators into a single framework. This way analysis and comparison of the different temporal aggregation forms becomes possible. Böhlen et al. [12] observe three limitations common to all existing temporal aggregation approaches. First, the temporal grouping process couples the partitioning of the timeline with the grouping of the input tuples. The time line is partitioned into intervals, and an input tuple is said to belong to a specific partition if its timestamp overlaps that partition. Second, the result tuples are defined for time points and not over time intervals. Third, they allow the use of at most one non-temporal attribute for temporal grouping.

The multi-dimensional temporal aggregation operator (TMDA) introduced in [12] overcomes these limitations and generalizes previous temporal aggregation operators. It decouples the partitioning of the timeline from the grouping of the input tuples, thus allowing one to specify result tuples over possibly overlapping intervals. Furthermore, the authors introduced fixed and constant interval semantics. Constant interval semantics compute the ITA, taking into consideration lineage information. With fixed interval semantics, which cover also the STA, the user specifies the time intervals for which to report result tuples. We use their definition of the ITA and respective algorithms as a starting point for the parsimonious temporal aggregation operator.

The approximation of temporal aggregation is a relatively new topic [22, 23, 53]. The work in [53] is the first to introduce an approximate temporal aggregation technique, which leverages from span temporal aggregation and, for a given time interval, finds an approximate aggregation result from tuples that overlap that interval. The approach uses off-the-shelf B- and R-trees to compute the aggregation result in linear space and logarithmic time with respect to the size of the database. Since the proposed technique approximates span temporal aggregation, where the user specifies the ag-

gregation intervals, it is not data-adaptive and cannot be used to reveal significant changes in the data. Moreover, only error-bounded approximation for the sum and $count$ aggregation functions is possible.

In [22, 23] we introduce parsimonious temporal aggregation as an approximation of ITA. In this thesis we extend our previous work in various directions. First, in addition to size-bounded PTA that reduces an ITA relation to a user-specified size, we define error-bounded PTA that minimizes the result size under a maximal error threshold specified by the user. We provide new query evaluation algorithms for error-bounded PTA, and we report the results of additional experiments, including the comparison of PTA with state of the art time series approximation techniques.

The work in [8] introduces an approximate temporal coalescing (ATC) technique to reduce the size of a temporal inverted file index. The index is represented as a temporal relation, where each record contains a document reference, a term, its index value, and a validity interval. ATC reads sorted and temporally adjacent tuples that share the same document/term pair and merges them if the induced local error does not exceed a user-specified threshold. Though the aim is different, ATC can be used to merge ITA result tuples. Experiments show that the total error of ATC is up to an order of magnitude higher than that of PTA and varies significantly depending on the dataset. Such a behavior is not surprising since ATC makes merging decisions based on local information only. The performance gain of ATC with respect to our greedy algorithms is negligible.

2.3 Histogram Construction

Histograms provide a compact synopsis of large datasets. Query optimizers, for example, use histograms to estimate query selectivity from histograms that capture attribute value distribution. Since the aim of PTA is to construct a compact synopsis of instant temporal aggregation, we may be able to leverage from solutions tailored for histogram construction. A good overview of the history of histogram development is provided in [26]. In this work we focus in particular on optimal histogram construction discussed primarily by Jagadish et al. [27]

In [27] the authors introduce an optimal algorithm that computes histogram representation of one dimensional data given either size or error bounds. Dynamic programming is used to obtain the optimal solution. The authors advocate the use of sum square error (SSE) measure and show how to evaluate it in constant time leading to $O(n^2c)$ time and $O(n^2)$ space complexity of the algorithm. They improve the performance even further by reducing the search space of the algorithm.

In this thesis we follow the above advice and use the same error measure. The dynamic programming algorithm for PTA evaluation introduced

here has been inspired by their approach. It is worth noting that although the algorithm in [27] considers only one dimensional data which has no equivalent of temporal gaps present in our setting, their search space reduction techniques nevertheless apply.

Recently a greedy algorithm for lattice histogram construction has been proposed by Karras [29]. The approach optimizes the error under infinite norm. With this error measure local decisions will lead to a globally optimal solution. Since we use a different error measure, the approach is not comparable to our work.

2.4 Time Series Approximation Methods

The techniques developed for approximate representation of time series data can benefit our research. An ITA result can be considered as time series if no temporal gaps and aggregation groups are present, and hence time series approximation techniques can be applied. ITA approximation is a more general problem whose subproblem is an approximation of time series data.

The need to visualize, mine and index abundant amounts of time series data has motivated an extensive research into their approximate representation. Many representation techniques have been proposed in the literature, such as [2, 40, 31, 61, 15, 41, 48]. An interesting classification of different approximation methods is available in [41]. Overall, the authors divide the methods into data adaptive and non data adaptive approaches. Wavelets, spectral methods, and piecewise aggregate approximation are examples of non-data adaptive approaches. Adaptive piecewise constant approximation and symbolic methods stand out as data adaptive approaches.

With respect to temporal aggregation, all of the time series approximation methods typically do not consider possible gaps in the data. Moreover, they compute the representation of one time series at a time, i.e. they approximate data from each aggregation group separately. Therefore, applying or comparing them in our setting is limited to such temporal data that does not sport any temporal gaps, and such temporal aggregation queries that do not specify aggregation groups.

Discrete wavelet transform (DWT) [51] was used for time series approximation in [40] and was shown to be superior to discrete Fourier transforms (DFT) [2]. The length of the input is assumed to be a power of two. Neighboring pairs are averaged obtaining a twice smaller representation of the input. Iterating recursively, DWT reaches the desired resolution of the data. In other words, the approach divides the input dataset into a set of equally long segments and, thus, is not data adaptive.

Piecewise aggregate approximation (PAA) was introduced in [31]. At

the same time the approach was introduced also in [61] and termed Segmenting Means. The approach divides the time series into c segments of equal lengths and calculates the average value within each segment. Just like DWT it is not a data adaptive approach. It has been shown that both PAA and DWT produce the same result under L_2 norm when the length of the input sequence and c are powers of 2. However, PAA is not limited to L_2 norm and the length of the input sequence. In fact, for any similarity norm L_p, a PAA approximation x' of a time series x with c segments is $\sqrt[p]{l}L_p(x') \leq L_p(x)$. Here l is a length of a segment, i.e. $l = \lceil n/c \rceil$.

Adaptive piecewise constant approximation (APCA) [15] leverages from PAA and DWT. It starts by decomposing the input time series using DWT. The wavelet coefficients are normalized, and only the c most significant coefficients are retained, which are then used to reconstruct the time series. Since the reconstruction step may yield up to $3c$ segments, the algorithm greedily merges the most similar ones to reduce the time series to size c. While APCA improves over PAA and DWT in terms of approximation quality, APCA induces significantly higher errors than our greedy evaluation algorithms. This is due to the underlying DWT decomposition, which is not data-adaptive and breaks apart the constant-value intervals in the ITA relation, yielding large approximation errors. The consequent greedy merging step of APCA can only smooth these errors out, but cannot fix them entirely.

The symbolic aggregate approximation SAX [41] and its scalable version iSAX [48] allow very efficient nearest neighbor queries by representing the input time series as a word of symbols. The symbolic representation is constructed in two steps. First, PAA is used to partition the time series into c equal sized segments. Second, the segments are represented using w different symbols in such a way that each symbol has approximately the same probability of occurrence. Increasing the vocabulary w leads to a more precise representations. Due to the use of PAA, the same limitations carry over to SAX.

In [18], an approach similar to ATC is proposed, which aims at guaranteeing a given error bound while maximizing the compression ratio of a continuous time series data stream. The proposed algorithm constructs a new approximation segment if attaching an incoming data point to the previous segment exceeds an error threshold. The data in each segment are approximated with a linear function. In line with other stream approximation techniques, the infinity norm is used as error measure, which allows to maintain the global approximation error under a given threshold by keeping local errors under the same threshold. However, the infinity norm is not appropriate for PTA, and we use the Euclidean norm as suggested in [27]. Additionally, our approach allows the user to specify either a global size or a global error bound.

In summary, time series approximation algorithms do not consider the constant value intervals in the ITA result. When applied in the context of

temporal aggregation, they split these intervals and yield high approximation errors. Moreover, temporal gaps in the data and the approximation of multiple aggregation groups under one global bound are not supported.

2.5 Topic Detection and Tracking

Topic Detection and Tracking (TDT) [44, 46, 14, 10, 1, 32, 3] is related to our problem of discovering aggregation groups or, in the context of news aggregation, discovery of evolving stories. The goal of TDT is to automatically identify topics within a set of documents, and to keep track of these topics as the document set evolves over time. Although the precise definition of what a topic is varies in terms of scope and generality across the literature, topics can be loosely understood to be large collections of documents related by a central theme, or belonging to a general category such as “Cinema” or “Entertainment”.

Topic detection can be online or retrospective [3]. In online setting new topics are discovered from a window of the most recent data. In retrospective setting, on the other hand, all the data collected up to the current time point is considered. Most existing TDT algorithms focus on the online discovery of new topics from recent data and the tracking of developments within known topics. Kleinberg [32] observed that the main problem in TDT is dealing with the tradeoff between content similarity and temporal locality. Piskorski et al. [46] suggest a sliding-window system where the data of the last four hours is clustered every ten minutes and overlapping clusters are subsequently linked into topics. Both [14, 10], suggest the discovery of topics from underlying stories by considering their cosine similarities within a seven day window. If the similarity between two stories is above a given threshold they are linked into a topic. Similarly, C-TREND [1], splits time into intervals and performs hierarchical clustering on events within each interval. Similar clusters from neighboring intervals are then linked. The algorithm is tailored towards trend visualization allowing efficient overview, zooming and filtering. For retrospective topic detection Allan et al. [3], suggest agglomerative clustering.

The critical difference between TDT and our approach is in the granularity of the clustering. We find that clusters in TDT are too general to represent individual stories evolving through time, and that a much finer clustering is needed. Indeed, Leskovec et al. [39], note that typical TDT topic clusters are too general to be of use in the analysis and understanding of the news cycle.

Besides generating more fine-grained clusters, our method differs from existing TDT algorithms in that it considers the entire time range for metastory detection. In other words, we merge related stories into metastories without any restriction on their temporal distance. Unlike [44, 3] we do not

use time decay in our similarity measure. In addition, we note that while standard TDT methods work on reasonably clean news document data, our initial story components contain blogs as well as news documents. Blogs are characterized by their large variability, and do not adhere to any standards either of content, language, or structure. This complicates the metastory generation process beyond what standard TDT was designed to handle.

2.6 Ranking

The temporal ranking framework that we propose in this thesis is closely related to PageRank [13] and HITS [33], two widely used approaches for ranking based on link analysis. PageRank exploits the linkage structure of a set of web pages using the intuition that the rank of a page should be high if it is linked-to by other highly ranked pages. Fast PageRank evaluation methods and an in-depth analysis are presented in [9, 38]. HITS [33] classifies web pages into hubs and authorities. Hubs are ranked high when they link to highly ranked authorities and, symmetrically, authorities receive high ranks when they are linked by qualitative hubs. PageRank and HITS cannot be directly applied to rank stories as there are no hyperlinks between them.

There have been a few attempts to rank individual documents that have few or no hyperlinks with the help of similarity scores [59, 35, 36, 37]. The authors of [59] address the problem of ranking sparsely linked web forum data where the links usually represent navigational aids or advertisements, but not recommendations for the user. In such a context, PageRank scores are uninformative. The authors propose exploiting the contents of the forum postings to influence the behavior of the random surfer. They hypothesize that the random surfer would most likely travel to nodes (postings) covering the same topic or its subtopics, and would rarely jump to a different topic altogether. They show that PageRank results improve when content-based implicit hyperlinks are introduced. In a similar fashion, Kritikopoulos et al. [35] suggest exploiting content similarity to supplement the sparse and often non-relevant link structure of weblog data. In contrast to these works, we assume no prior hyperlinks between stories. The transition probability matrix that we use for ranking is computed using only the selected similarity function.

Kurland et al. [36, 37] suggest PageRank and HITS style algorithms to rank documents based on their content similarity. Asymmetric Kullback-Leibler divergence is used to compare document keyword vectors yielding a weighted directed flow graph which, subsequently, is fed to the PageRank algorithm. The authors apply HITS by constructing a bipartite graph from document clusters and individual documents. The clusters are treated as

hubs, whereas individual documents are treated as authorities. They use only resulting authority scores for the final ranking and show that the HITS approach outperforms PageRank.

In contrast to their work, we argue that using a symmetric similarity measure for metastory components is more intuitive and appropriate. Furthermore, the symmetric property of the similarity measure allows very efficient evaluation of PageRank. In our experimental evaluation we compare PageRank used in our framework against the ranking results produced by the HITS algorithm, evaluating in addition the impact of different similarity measures on the ranking result.

2.7 Summary

In this chapter we have reviewed the most recent and prominent works in various research fields that relate to the problem of this thesis. We have discussed various temporal aggregation operators and corresponding evaluation algorithms that form the groundwork for parsimonious temporal aggregation. We have looked at histogram construction and time series approximation algorithms. We will leverage from these works when creating dynamic programming and greedy evaluation strategies for the new temporal aggregation operator. Finally, we have discussed topic detection and tracking as well as ranking problems that relate closely to our temporal ranking framework. In the next chapter we use this knowledge to define the parsimonious temporal aggregation operator.

CHAPTER 3

Parsimonious Temporal Aggregation

Temporal aggregation is an important operation in temporal databases, and different variants thereof have been proposed. In this chapter we introduce a novel temporal aggregation operator, termed parsimonious temporal aggregation (PTA), which overcomes major limitations of existing approaches. PTA takes the result of instant temporal aggregation (ITA) of size n — which might be up to twice as large as the argument relation — and merges similar tuples until a given error (ϵ) or size (c) bound is reached. The new operator is data-adaptive and allows the user to control the tradeoff between the result size and the error induced by merging. For the precise evaluation of PTA queries, we propose two dynamic programming based algorithms for size- and error-bounded queries, respectively, with a worst-case complexity that is quadratic in n. We present two optimizations that take advantage of temporal gaps and different aggregation groups and achieve a linear runtime in experiments with real-world data. Empirical study using synthetic and real world data reveals the usefulness of PTA. The operator considerably and consistently reduces the aggregation result introducing only small errors.

3.1 Introduction

Temporal aggregation is used to summarize large sets of such data and is, therefore, very important in temporal databases. It has been studied in various flavors, most importantly, as instant, and span temporal aggregation. However, current aggregation operators may not meet the demands of modern data mining applications as they do not allow control over the size of the output.

In instant temporal aggregation (ITA) [12, 34, 43, 55, 42, 60], the aggregate value at a time instant t is computed from the set of tuples whose timestamps overlap t. Consecutive time instants with identical aggregate values can then be coalesced into so-called constant intervals, i.e., tuples valid over maximal time intervals during which the aggregate results are constant. Nevertheless, the result size depends on the argument relation and, due to temporally overlapping argument tuples, may become up to twice the size of the input [12]. This behavior is in conflict with the very idea of aggregation, which is to provide a summary of the data.

On the contrary, span temporal aggregation (STA) [12, 34] allows an application to specify the time intervals for which to report result tuples, e.g., for each year from 2000 to 2005. For each of these intervals a result tuple is produced by aggregating all argument tuples that overlap that interval. Therefore, the output size of STA is predictable yet it may fail to provide good summaries of the data since the aggregation intervals do not consider the distribution of the underlying data.

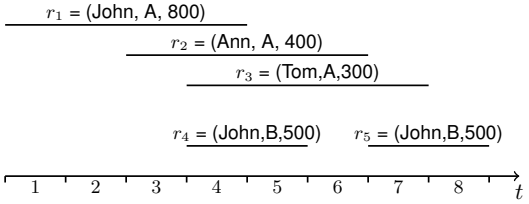
In this chapter we introduce the parsimonious temporal aggregation (PTA) operator that combines the best features of instant and span temporal aggregation. It aims at providing the user with a small set of result tuples that represent the most significant changes in the data over time. Like ITA, PTA is data adaptive as it considers the (temporal) distribution of the input. At the same time, similarly to STA, it allows one to control the size of the result. Conceptually PTA operates in two steps: (1) it computes the ITA result of the input relation and, (2) it reduces it by merging the temporally adjacent ITA result tuples and keeping the introduced error minimal until a user-specified size or error bound is satisfied. Tuples are adjacent if they belong to the same aggregation group and are not separated by a temporal gap. PTA is useful for such applications as data visualization or similarity search for classification and clustering where the fine-grained result of ITA is too large to handle and, instead, a concise overview of the data at hand is necessary.

In particular, we formally define the PTA operator for two types of user queries. Size-bounded PTA reduces an ITA relation to a user-specified size minimizing the global error. Error-bounded PTA minimizes the result size under a user specified global error threshold. We introduce a dynamic programming algorithm tailored for offline evaluation of PTA queries. It finds

the optimal reduction of the ITA relation for both, size and error-bounded queries. We show that aggregation groups specified in the query and temporal gaps present in the data allow significant performance improvement yielding almost linear running times when evaluated using real-world datasets. In the worst case the algorithm runs in $O(n^2cp)$ time and needs $O(n^2)$ space. Finally, we conduct extensive experimental evaluation using real-world as well as synthetic data. The results show that the PTA operator allows a significant reduction of the ITA result while introducing only a small error.

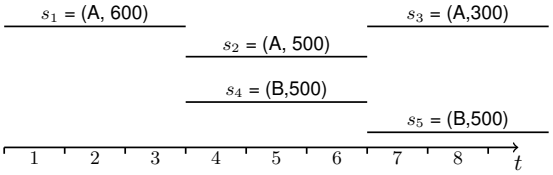
3.2 Example

	Empl	*Proj*	*Sal*	*T*
r_1	John	A	800	[1, 4]
r_2	Ann	A	400	[3, 6]
r_3	Tom	A	300	[4, 7]
r_4	John	B	500	[4, 5]
r_5	John	B	500	[7, 8]



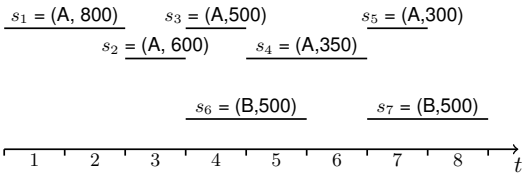
(a) **proj** Relation

	Proj	*AvgSal*	*T*
s_1	A	500	[1, 4]
s_2	A	350	[5, 8]
s_3	B	500	[1, 4]
s_4	B	500	[5, 8]



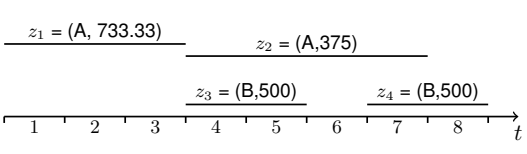
(b) STA Result

	Proj	*AvgSal*	*T*
s_1	A	800	[1, 2]
s_2	A	600	[3, 3]
s_3	A	500	[4, 4]
s_4	A	350	[5, 6]
s_5	A	300	[7, 7]
s_6	B	500	[4, 5]
s_7	B	500	[7, 8]



(c) ITA Result

	Proj	*AvgSal*	*T*
z_1	A	733.33	[1, 3]
z_2	A	375	[4, 7]
z_3	B	500	[4, 5]
z_4	B	500	[7, 8]



(d) PTA Result of Size 4

Figure 3.1: Temporal aggregation over the **proj** relation using the query *"What is the average monthly salary for each project?"*

Throughout the chapter we use the relation **proj** shown in Figure 3.1(a) to illustrate and explain the concepts. It records the information about project contracts and each record stores the name of an employee ($Empl$), the project he/she works for ($Proj$), the monthly salary (Sal), and the time period (T) (in months) during which the contract is effective. Thus the tuple r_1 states that John received a salary of 800 for each of the four months that he worked on the project A. In the graphical illustration the timestamps of the tuples are drawn as horizontal lines.

Consider a STA query *"For each project, what is the average monthly salary in each trimester?"*. It implicitly specifies the expected maximal result size. Evaluation of the query involves constructing one result tuple per each 4 month period and aggregation group as shown in Figure 3.1(b). Since the aggregation intervals are imposed over the data, the approach is likely to miss important changes that do not dominate the interval.

Figure 3.1(c) shows the result of an ITA query *"For each project, what is the average monthly salary?"*. Evaluation involves computing the average salary at each time point for both aggregation groups, projects A and B, from the tuples that hold at that point. Then identical values over consecutive time points are then coalesced. The result contains one grouping and one aggregation attribute. Although its size exceeds the size of the input we observe that some adjacent tuples have quite similar aggregate values, e.g., s_4 and s_5.

Figure 3.1(d) shows the result of a PTA query *"For each project, what is the average monthly salary, where the result size shall not exceed 4 tuples?"*. The result is obtained by applying three merging steps such that the total error between the reduced relation and the ITA relation is minimal. The ITA tuples s_1 and s_2 are merged to the PTA tuple z_1. The average salary of z_1 is computed by averaging the salaries of s_1 and s_2 over each time point, i.e., 800 for two months and 600 for one month, yielding the value 733.33. The tuples separated by temporal gaps, e.g. s_6 and s_7, or belonging to different aggregation groups, e.g. s_3 and s_6, are not merged. In contrast to the other operators the PTA reveals significant changes in the aggregation values.

3.3 Preliminaries

3.3.1 Temporal Database Model

First we formally define the temporal database model used throughout this thesis. Let $\mathbf{\Omega}$[1] denote a non-empty, finite set of attributes. Let β denote a finite set of domains, and $dom : \mathbf{\Omega} \rightarrow \beta$ be a function that associates a domain with each attribute. We call a triple $R = (\Omega, \beta, dom)$ a *schema* and

[1] Boldface is used to denote sets throughout.

use it to describe the format of data tuples. A *tuple* r *over schema* R is a finite set that contains for every $A_i \in \Omega$ a pair A_i/v_i such that $v_i \in dom(A_i)$. A *relation over schema* R is a finite set of tuples over R.

A *temporal schema* is a schema with at least one timestamp valued attribute, T, ranging over *time domain* β^T, i.e., $T \in \Omega$ and $dom(T) = \beta^T \in \beta$. We assume that the time domain is discrete. Its elements are termed chronons (or time points), equipped with a total order, $<^T$ (e.g., calendar months with the usual chronological order). A *timestamp* (or time interval), $\mathbf{t}$, is a convex set of chronons over the time domain and is represented by two chronons, $[t_b, t_e]$, denoting its inclusive starting and ending points, respectively. If $\mathbf{t} \cap \mathbf{t}' \neq \emptyset$, we say that the two intervals overlap (or intersect), otherwise, we say that they are disjoint.

For simplicity we assume an ordering of the attributes and represent a temporal relation schema as $R = (A_1, \ldots, A_m, T)$ and a corresponding tuple as $r = (v_1, \ldots, v_m, \mathbf{t})$. For a tuple r and an attribute A we write $r.A$ to denote the value of the attribute A in r, thus $r.T = \mathbf{t}$. For a set of attributes $\mathbf{A} = \{A_1, \ldots, A_k\}$, $k \leq m$, we define $r.\mathbf{A} = (r.A_1, \ldots, r.A_k)$.

3.3.2 Instant Temporal Aggregation

ITA computes an aggregation result for each combination of grouping attribute values, at each time point, t, by considering all argument tuples that hold at t and have the same grouping attribute values. Formally,

DEFINITION 3.1 (Instant Temporal Aggregation)**.** *Let* $\mathbf{r}$ *be a temporal relation with schema* $R = (A_1, \ldots, A_m, T)$*, grouping attributes* $\mathbf{A} = \{A_1, \ldots, A_k\}$*, and aggregate functions* $\mathbf{F} = \{f_1/B_1, \ldots, f_p/B_p\}$*. Furthermore, let coalesce be the coalescing operator and* $\mathbf{r}_{g,t} = \{r \mid r \in \mathbf{r} \wedge r.\mathbf{A} = g \wedge t \in r.T\}$ *be all tuples of* $\mathbf{r}$ *with grouping attribute values equal to* g *and intersecting time point* t*.* Instant temporal aggregation *is defined as*

$$\mathcal{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{r} = coalesce\{s \mid g \in \pi[\mathbf{A}]\mathbf{r} \wedge t \in \beta^T \wedge \mathbf{r}_{g,t} \neq \emptyset \wedge \tag{3.1}$$

$$f = (f_1(\mathbf{r}_{g,t}), \ldots, f_p(\mathbf{r}_{g,t})) \wedge \tag{3.2}$$

$$s = g \circ f \circ [t, t]\}. \tag{3.3}$$

and has schema $S = (A_1, \ldots, A_k, B_1, \ldots, B_p, T)$*.*

In the above, the variable g ranges over all combinations of grouping attribute values in $\mathbf{r}$, and the variable t ranges over the time domain. For each combination of g and t, the aggregation group $\mathbf{r}_{g,t}$ collects all argument tuples that have grouping attribute values equal to g and are valid at time t. A result tuple, s, is produced by extending g with the result of the aggregate functions f_i evaluated over the non-empty $\mathbf{r}_{g,t}$ and with a timestamp $[t, t]$. Each f_i is some aggregation function that takes a (temporal) relation as argument and applies aggregation to one of the relation's attributes. The

resulting value is stored as the value of an attribute named B_i. The final step is coalescing of value-equivalent result tuples over consecutive time points into tuples over maximal time periods during which the aggregate values do not change. The result of ITA contains up to $2n-1$ tuples, where n is the size of the argument relation [12].

Example 3.1. The ITA query *"What is the average monthly salary for each project?"* in our running example (see also Figure 3.1(c)) is formulated as $\mathcal{G}^{ITA}[\mathbf{A}, \mathbf{F}]$**proj** with aggregate functions $\mathbf{F} = \{avg(Sal)/AvgSal\}$ and grouping attributes $\mathbf{A} = \{Proj\}$. The aggregation result is relation with schema $(Proj, AvgSal, T)$.

A property of ITA aggregation result relation is that the timestamps of the tuples within a single aggregation group do not intersect. We term such temporal relations *sequential*.

DEFINITION 3.2. (Sequential Relation) *A relation* s *over a temporal relation schema* $S = (A_1, \ldots, A_m, T)$ *is sequential with respect to a set of (grouping) attributes* $\mathbf{A} \subseteq \{A_1, \ldots, A_m\}$ *if for any pair of tuples* $s_i, s_j \in \mathrm{s}$ *such that* $s_i \neq s_j \wedge s_i.\mathbf{A} = s_j.\mathbf{A}$ *we have* $s_i.T \cap s_j.T = \emptyset$.

For example, the ITA result in Figure 3.1(c) is sequential with respect to the grouping attribute $Proj$, i.e., the timestamps of all tuples with identical $Proj$ values are temporally disjoint. There are many other instances of sequential relations, e.g., time series datasets.

3.4 Parsimonious Temporal Aggregation

In this section we introduce and define parsimonious temporal aggregation, PTA, which conceptually comprises two steps: (1) obtain the ITA result from the argument relation and, (2) merge adjacent ITA result tuples until a user specified size or error bound is satisfied. We begin by describing the merging of adjacent tuples and an error measure that is used to quantify the induced error.

3.4.1 Merging Adjacent Tuples

The ITA result is always a sequential relation, which shall be preserved by allowing only adjacent tuples to be merged.

DEFINITION 3.3 (Adjacent Tuples)**.** *Let* s *be a sequential relation with schema* $S = (A_1, \ldots, A_k, B_1, \ldots, B_p, T)$ *and grouping attributes* $\mathbf{A} = \{A_1, \ldots, A_k\}$.

Two tuples $s_i, s_j \in \mathbf{s}$ *are* adjacent*,* $s_i \prec s_j$*, iff the following holds:*

$$(1) \quad s_i.\mathbf{A} = s_j.\mathbf{A} \tag{3.4}$$

$$(2) \quad s_i.t_e = s_j.t_b - 1 \tag{3.5}$$

$$\tag{3.6}$$

The first condition ensures that the two tuples are value-equivalent in the non-temporal attributes. The second condition requires that the tuples are immediately consecutive and not separated by a temporal gap.

Example 3.2. In the ITA result in Figure 3.1(c) we have $s_1 \prec s_2 \prec s_3 \prec s_4 \prec s_5$. Tuples s_5 and s_6 are not adjacent, $s_5 \not\prec s_6$, since the $Proj$-values are different, violating the first condition. Similar, s_6, s_7 and s_1, s_3 are not adjacent since they are separated by a temporal gap and violate the second condition.

DEFINITION 3.4 (Merge Operator)**.** *Let* $\mathbf{s}$ *be an ITA result relation with schema* $S = (A_1, \ldots, A_k, B_1, \ldots, B_p, T)$*, where* $\mathbf{A} = \{A_1, \ldots, A_k\}$ *are the grouping attributes and* $\mathbf{B} = \{B_1, \ldots, B_p\}$ *store the aggregate values. The* merge*,* $\oplus$*, of two adjacent tuples,* $s_i, s_j \in \mathbf{s}$*,* $s_i \prec s_j$*, is defined as*

$$s_i \oplus s_j = (s_i.A_1, \ldots, s_i.A_k, v_1, \ldots, v_p, [s_i.t_b, s_j.t_e]), \tag{3.7}$$

where $v_d = \frac{|s_i.T|s_i.B_d + |s_j.T|s_j.B_d}{|s_i.T| + |s_j.T|}$ *for* $1 \leq d \leq p$*.*

The merge operator produces a new tuple from two ITA result tuples, i.e., $z = s_i \oplus s_j$. The grouping attributes of z assume identical values as s_i (and s_j). The timestamp $z.T$ is the concatenation of s_i's and s_j's timestamp. Since the aggregate values of s_i and s_j hold at every time point in $s_i.T$ and $s_j.T$, respectively, the new aggregate values, $v_1, \ldots, v_p$, are computed by averaging over the timestamps, i.e., v_d is the weighted average of $s_i.B_d$ and $s_j.B_d$ with the weights being the length of $s_i.T$ and $s_j.T$, respectively.

Example 3.3. Merging the tuples $s_1 = (A, 800, [1, 2])$ and $s_2 = (A, 600, [3, 3])$ in Figure 3.1(c) yields the result tuple $z_1 = s_1 \oplus s_2 = (A, 733.33, [1, 3])$ in Figure 3.1(d). The average salary is determined as $z_1.AvgSal = (2 \cdot 800 + 1 \cdot 600)/(2 + 1) = 733.33$.

To reduce the ITA result, $\mathbf{s}$, to a specific size, the merge operator is applied recursively. However, there is a *lower bound*, c_{min}, for the size of the reduced ITA result, which is determined by the difference between the cardinality of $\mathbf{s}$ and the number of adjacent tuple pairs that can be merged, i.e., $c_{min} = |\mathbf{s}| - |\{(s_i, s_j)|s_i, s_j \in \mathbf{s} \wedge s_i \prec s_j\}|$. In our running example, the ITA result contains seven tuples with four adjacent pairs, giving $c_{min} = 7 - 4 = 3$.

Next, we introduce a (nondeterministic) reduction function that reduces an ITA result relation to a given size c.

DEFINITION 3.5 (Reduction Function). *Let* s *be an ITA result relation,* $s_i \prec s_j$ *be two adjacent tuples in* s*, and* $c \geq c_{min}$ *be a size constraint. The* reduction*,* ρ*, of relation* s *to size* c *is defined as*

$$\rho(\mathbf{s}, c) = \begin{cases} \mathbf{s} & |\mathbf{s}| \leq c, \\ \rho(\mathbf{s} \setminus \{s_i, s_j\} \cup \{s_i \oplus s_j\}, c) & |\mathbf{s}| > c. \end{cases} \tag{3.8}$$

If the cardinality of s is smaller or equal to c, the reduction process terminates. Otherwise, two adjacent tuples, s_i and s_j, are substituted with the merged tuple $s_i \oplus s_j$. Notice the nondeterministic nature of ρ which allows any pair of adjacent tuples to be merged. We will be more specific about choosing tuples for merging later on.

Example 3.4. The ITA result relation in Figure 3.1(c) is reduced to size $c = 4$ in three merging steps with $\rho(\mathbf{s}, 4)$. The reduced relation in Figure 3.1(d) is obtained by merging tuples s_1, s_2 into z_1 and $s_3 \oplus (s_4 \oplus s_5)$ into z_2. Choosing different pairs of tuples for merging produces different results.

3.4.2 The Error Measure

Merging tuples induces an error with respect to the ITA result, which we quantify using the following error measure.

DEFINITION 3.6 (Error Measure). *Let* $\mathbf{s}, S, \mathbf{A}, \mathbf{B}$ *be as in Def. 3.4,* $\mathbf{z} = \rho(\mathbf{s}, \cdot)$ *be a reduction of* $\mathbf{s}$*, and let for each* $z \in \mathbf{z}$*,* $\mathbf{s}_z = \{s \mid s \in \mathbf{s} \wedge s.\mathbf{A} = z.\mathbf{A} \wedge s.T \subseteq z.T\}$ *be the set of all ITA result tuples that are merged into* z*. For a set of positive weights,* $w_1 > 0, \ldots, w_p > 0$*, the* error*,* $SSE(\mathbf{s}, \mathbf{z})$*, that is induced by reducing* $\mathbf{s}$ *to* $\mathbf{z}$ *is*

$$SSE(\mathbf{s}, \mathbf{z}) = \sum_{z \in \mathbf{z}} \sum_{s \in \mathbf{s}_z} \sum_{d=1}^{p} w_d^2 |s.T| (s.B_d - z.B_d)^2. \tag{3.9}$$

This is the well-known sum squared error, which is given as the total sum of the squared distance between the tuples in s and $\mathbf{z}$. More specifically, it computes for each tuple, $z \in \mathbf{z}$, the squared distance (over all aggregation results, $B_1, \ldots, B_p$) between z and the ITA result tuples, $s \in \mathbf{s}_z$, that are merged to produce z. The weights w_d are used to leverage the impact of the different aggregation attributes. The choice of such weights is out of the scope of this work, and we refer the interested reader to [58].

Example 3.5. Consider the merge of tuples $s_1 = (A, 800, [1, 2])$ and $s_2 = (A, 600, [3, 3])$ in Figure 3.1(c) to tuple $z = (A, 733.33, [1, 3])$ in Figure 3.1(d). With a weight of 1 for the only aggregation attribute $AvgSal$, the induced error is $SSE(\mathbf{s}, \mathbf{z}) = 1 \cdot 2 \cdot (800 - 733.33)^2 + 1 \cdot 1 \cdot (600 - 733.33)^2 = 26\,666.67$.

3.4.3 The PTA Operator

We provide two variants of the PTA operator. First, size-bounded PTA reduces the ITA relation to a user-specified size, yet minimizing the induced error. Second, error-bounded PTA reduces the the size of the ITA result relation as much as possible, yet maintaining the total induced error under a given threshold.

DEFINITION 3.7 (Size-Bounded PTA). *Let* $\mathbf{r}$ *be a temporal relation with schema* $R = (A_1, \ldots, A_m, T)$*, grouping attributes* $\mathbf{A} = \{A_1, \ldots, A_k\}$*, and aggregate functions* $\mathbf{F} = \{f_1/B_1, \ldots, f_p/B_p\}$*. Let* $c \geq c_{min}$ *be an application-specific size constraint. A relation* $\mathbf{z}$ *is the result of* size-bounded parsimonious temporal aggregation*,* $\mathbf{z} = \mathcal{G}^{PTA}[\mathbf{A}, \mathbf{F}, c]\mathbf{r}$*, iff*

$$(1) \quad \mathbf{s} = \mathcal{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{r} \tag{3.10}$$

$$(2) \quad \mathbf{z} = \rho(\mathbf{s}, c) \tag{3.11}$$

$$(3) \quad \nexists \mathbf{z}'(\mathbf{z}' = \rho(\mathbf{s}, c) \wedge SSE(\mathbf{s}, \mathbf{z}') < SSE(\mathbf{s}, \mathbf{z})). \tag{3.12}$$

Relation $\mathbf{s}$ is the ITA result which is reduced to c tuples in the best possible way, that is, there is no better reduction, $\mathbf{z}'$, of $\mathbf{s}$ to $c' < c$ tuples that would induce a smaller error. An PTA result is not necessarily unique. If different reductions to size c induce the same minimal error, all of them represent valid PTA results.

Example 3.6. There are four different ways to reduce the ITA result in Figure 3.1(c) to $c = 4$ tuples. Figure 3.1(d) shows the best possible reduction with an induced error of 49 166. Figure 4.2 shows a different reduction, which has an error of 63 000.

DEFINITION 3.8 (Error-Bounded PTA). *Let* $\mathbf{r}$*,* R*,* $\mathbf{A}$*, and* $\mathbf{F}$ *be as in Def. 3.7 and* ϵ*,* $0 \leq \epsilon \leq 1$*, be an application-specific error bound. Furthermore, let* $SSE_{max} = SSE(\mathbf{s}, \rho(\mathbf{s}, c_{min}))$ *denote the largest possible error. A relation* $\mathbf{z}$ *is the result of* error-bounded parsimonious temporal aggregation*,* $\mathbf{z} = \mathcal{G}^{PTA}[\mathbf{A}, \mathbf{F}, \epsilon]\mathbf{r}$ *iff*

$$(1) \quad \mathbf{s} = \mathcal{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{r} \tag{3.13}$$

$$(2) \quad \exists c(\mathbf{z} = \rho(\mathbf{s}, c)) \tag{3.14}$$

$$(3) \quad SSE(\mathbf{s}, \mathbf{z}) \leq \epsilon \cdot SSE_{max} \tag{3.15}$$

$$(4) \quad \nexists \mathbf{z}', c'(\mathbf{z}' = \rho(\mathbf{s}, c') \wedge c' \leq c \wedge SSE(\mathbf{s}, \mathbf{z}') < SSE(\mathbf{s}, \mathbf{z})). \tag{3.16}$$

Relation $\mathbf{z}$ is a maximal reduction of $\mathbf{s}$ (to a size c) such that the induced error is smaller or equal to ϵ multiplied by the largest possible error, which occurs when $\mathbf{s}$ is reduced to c_{min} tuples. Condition 4 ensures that no reduction to a smaller or same number, $c' \leq c$, of tuples exists with a smaller error. Again, the result may not be unique.

Example 3.7. With an error threshold $\epsilon = 1$ we obtain obviously the maximal reduction of two tuples. Allowing only 2% error yields four result tuples as in Figure 3.1(d).

3.5 PTA Evaluation Using Dynamic Programming

For the evaluation of PTA queries, ITA is computed first, followed by a reduction of the ITA result until a given size or error bound is satisfied. In this section we propose algorithms $\mathrm{PTA_c}$ and PTA_ϵ for the precise evaluation of size-bounded and error-bounded PTA, respectively. While any ITA algorithm can be used for the first step, a dynamic programming based approach is adopted to compute an optimal reduction of the ITA result. Various optimizations are applicable to improve the basic DP scheme, such as computing the error in constant time and exploiting temporal gaps and aggregation groups to prune the search space.

3.5.1 Basic DP Scheme for Size-Bounded PTA

Let $\mathrm{s} = \{s_1, \ldots, s_n\}$ be an ITA result relation sorted on the aggregation groups and, within each aggregation group, along the timeline. Then each pair of consecutive tuples that are non-adjacent, $s_i \not\prec s_{i+1}$, marks a boundary (temporal gap or change of aggregation group) that cannot be crossed during the merging process.

Example 3.8. Consider the ITA result in Figure 3.1(c), which is sorted first by the $Proj$ attribute and, within each group, in chronological order. It contains two boundaries, namely s_5, s_6 since the two tuples belong to different aggregation groups, and s_6, s_7 since the two tuples are separated by a temporal gap.

Let $\mathrm{s}_j = \{s_1, \ldots, s_j\}$ denote the first j tuples in s. Then $\mathrm{s} \backslash \mathrm{s}_j$ denotes the rest of the relation, i.e. $\mathrm{s} \backslash \mathrm{s}_j = \{s_{j+1}, \ldots, s_n\}$ the rest. Then the reduction of s to c tuples, $\rho(\mathrm{s}, c)$, can be defined recursively as follows: find a reduction $\rho(\mathrm{s}_j, c-1)$ for some *split point* j and merge the remaining tuples into one, i.e., $\rho(\mathrm{s} \setminus \mathrm{s}_j, 1)$. For the reduction to be optimal, the sum of errors induced on both sides of j must be minimized at each recursive step. To avoid that non-adjacent tuples are merged, we set the error of merging non-adjacent tuples to infinity. Thus, merging altogether any subset $\mathrm{s}' \subseteq \mathrm{s}$ will yield infinite error, if s' contains at least one pair of non-adjacent tuples, $s_i \not\prec s_{i+1}$.

Example 3.9. Figure 3.2 illustrates the four options for the split point, j. For instance, for $j = 3$, the solution is to find an optimal reduction $\rho(\mathrm{s}_3, 3)$ and to merge s_4, s_5, s_6, s_7 into one tuple, which yields an infinite error since $s_5 \not\prec s_6 \not\prec s_7$. The only split point with an error different from ∞ is $j = 6$.

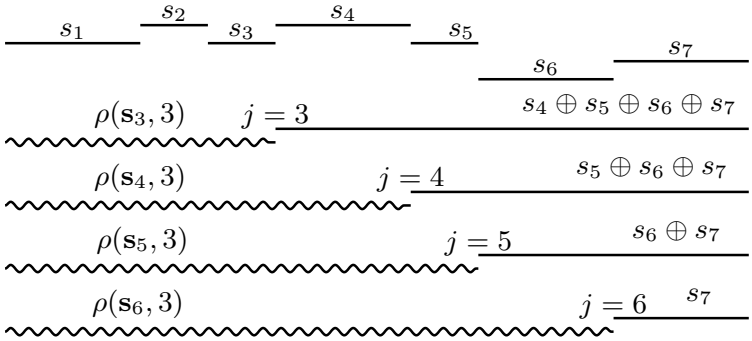


Figure 3.2: Four possible ways to reduce the ITA result to four tuples

	$i=1$	2	3	4	5	6	7
$k=1$	0	26666	67500	208333	269285	∞	∞
2	–	0	5000	41666	49166	269285	∞
3	–	–	0	5000	6666	49166	269285
4	–	–	–	0	1666	6666	49166

Figure 3.3: Error matrix $\mathbf{E}$

To find an optimal reduction, we propose a dynamic programming technique that constructs an an *error matrix*, $\mathbf{E}_{c \times n}$, with c rows and n columns. A cell (k, i) represents the smallest error of reducing $\mathbf{s}_i$ to k tuples. The matrix is filled incrementally in each step using the values that have been computed in the previous steps, i.e., $\mathbf{E}_{k,i} =$

$$\begin{cases} \min\limits_{k-1 \leq j < i} \{\mathbf{E}_{k-1,j} + \\ \qquad SE(\mathbf{s}_i \backslash \mathbf{s}_j, \rho(\mathbf{s}_i \backslash \mathbf{s}_j, 1))\} & k > 1 \\ SE(\mathbf{s}_i, \rho(\mathbf{s}_i, 1)) & k = 1 \wedge s_1 \prec \ldots \prec s_i \\ \infty & k = 1 \wedge \neg(s_1 \prec \ldots \prec s_i) \end{cases} \tag{3.17}$$

The matrix is filled row-wise for all $k = 1, \ldots, c$, and, for any fixed k, in increasing order of i for $i = 1, \ldots, |\mathbf{s}|$. At each step k, the values computed in steps $k-1$ are used. At the end, the value $\mathbf{E}_{c,n}$ contains the error induced by an optimal reduction of relation $\mathbf{s}$ to c tuples.

Example 3.10. Figure 3.3 shows the error matrix that is constructed when reducing the ITA result in our running example to size 4. The matrix is filled starting from row $k = 1$. To fill the second row, the data from row 1 are used, etc. Eventually, cell $(4, 7)$ contains the error of the optimal reduction.

In order to construct the reduced relation, we maintain a *split point matrix*, $\mathbf{J}_{c \times n}$. A cell (k, i) in the matrix stores the value of j that led to the min-

	$i=1$	2	3	4	5	6	7
$k=1$	0	[0]	0	0	0	0	0
2	–	1	1	2	[2]	5	0
3	–	–	2	3	3	[5]	6
4	–	–	–	3	3	5	[6]

Figure 3.4: Split point matrix $\mathbf{J}$

imal error value when computing $\mathbf{E}_{k,i}$. Consequently the cell (c, n) stores the first split point j that tells us where to split $\mathbf{s}$ in order to construct the final result. The tuples $s_{j+1}, \ldots, s_n$ are then merged into a single one, whereas tuples $s_1, \ldots, s_j$ are merged into $c-1$ tuples, following the next split point that is stored in the cell $(c-1, j)$, etc.

Example 3.11. Figure 3.4 shows the split point matrix in our running example. The split points of the optimal reduction are framed. The first split point is $j = 6$, the value of cell $(4, 7)$. We generate the result tuple $z_4 = s_7$ and proceed to reduce $\mathbf{s}_6$ to size 3 by taking the value of cell $(3, 6)$ as the next split point. We generate the result tuple $z_3 = s_6$. Then we proceed to reduce $\mathbf{s}_5$ to two tuples, obtaining $z_2 = s_3 \oplus s_4 \oplus s_5$. Finally, we reduce $\mathbf{s}_2$ to size 1, yielding $z_1 = s_1 \oplus s_2$ (the last split point in cell $(2, 1)$ is 0).

3.5.2 Efficient Computation of the Error

The DP scheme frequently needs to compute the error that is induced when a set of adjacent tuples is merged. Jagadish et al. [27] introduce a technique to calculate the error for one dimensional data in constant time. We extend their approach for multidimensional data.

Let $\mathbf{s}$ be an ITA result relation with corresponding aggregation attributes $\mathbf{B} = \{B_1, \ldots, B_p\}$. The additional information that is required for an efficient computation of the error is stored in two matrices $\mathbf{S}_{p\times|\mathbf{s}|}$, $\mathbf{SS}_{p\times|\mathbf{s}|}$ and a vector $\mathbf{L}_{|\mathbf{s}|}$, which are defined as follows:

$$\mathbf{S}_{d,i} = \sum_{j=1}^{i} |s_j.T| s_j.B_d, \tag{3.18}$$

$$\mathbf{SS}_{d,i} = \sum_{j=1}^{i} |s_j.T| s_j.B_d^2, \tag{3.19}$$

$$\mathbf{L}_i = \sum_{j=1}^{i} |s_j.T|. \tag{3.20}$$

$\mathbf{S}_{d,i}$ is the sum of the B_d values over all tuples from s_1 to s_i, $\mathbf{SS}$ is the sum of the squares of the B_d values, and $\mathbf{L}$ is the sum of the lengths of the timestamps. Observe that precomputing this information does not induce any additional overhead since the ITA algorithm can fill the matrices while producing the output.

Using such information, the error of merging a set of ITA result altogether can be computed in $O(p)$ time, where p is the number of aggregation attributes in the ITA result. This is shown in the following proposition.

PROPOSITION 3.1. *Let $\mathbf{s}_z = \{s_i, s_{i+1}, \ldots, s_j\} \subseteq \mathbf{s}$ such that $s_i \prec \ldots \prec s_j$. The error that is induced by merging $\mathbf{s}_z$ into one tuple, z, can be computed as*

$$SSE(\mathbf{s}_z, \{z\}) = \sum_{d=1}^{p} w_d^2 \left[\mathbf{SS}_{d,j} - \mathbf{SS}_{d,i-1} - \frac{(\mathbf{S}_{d,j} - \mathbf{S}_{d,i-1})^2}{\mathbf{L}_j - \mathbf{L}_{i-1}} \right]. \tag{3.21}$$

Proof. From Def. 3.4 of the merge operator we have that

$$z.B_d = \frac{1}{|z.T|} \sum_{s \in \mathbf{s}_z} |s.T| s.B_d. \tag{3.22}$$

Then we rewrite the error equation in Def. 3.6 as follows:

$$SSE(\mathbf{s}_z, \{z\}) = \sum_{d=1}^{p} w_d^2 \Bigg[\sum_{s \in \mathbf{s}_z} |s.T| s.B_d^2 - \tag{3.23}$$

$$2\, z.B_d \underbrace{\sum_{s \in \mathbf{s}_z} |s.T| s.B_d}_{=|z.T| z.B_d} + z.B_d^2 \underbrace{\sum_{s \in \mathbf{s}_z} |s.T|}_{=|z.T|} \Bigg] \tag{3.24}$$

$$= \sum_{d=1}^{p} w_d^2 \left[\sum_{s \in \mathbf{s}_z} |s.T| s.B_d^2 - |z.T|\, z.B_d^2 \right] \tag{3.25}$$

$$= \sum_{d=1}^{p} w_d^2 \left[\sum_{s \in \mathbf{s}_z} |s.T| s.B_d^2 - \frac{(\sum_{s \in \mathbf{s}_z} |s.T| s.B_d)^2}{|z.T|} \right] \tag{3.26}$$

$$= \sum_{d=1}^{p} w_d^2 \left[\mathbf{SS}_{d,j} - \mathbf{SS}_{d,i-1} - \frac{(\mathbf{S}_{d,j} - \mathbf{S}_{d,i-1})^2}{\mathbf{L}_j - \mathbf{L}_{i-1}} \right]. \tag{3.27}$$

□

Example 3.12. For the ITA result in Figure 3.1(c) the matrices and vectors are given as follows:

$$\begin{aligned}
\mathbf{S} &= \langle\ 1\,600,\ 2\,200,\ 2\,700,\ 3\,400,\ \ldots\ \rangle \\
\mathbf{SS} &= \langle\ 1\,280\,000,\ 1\,640\,000,\ 1\,890\,000,\ 2\,135\,000,\ \ldots\ \rangle \\
\mathbf{L} &= \langle\ 2,\ 3,\ 4,\ 6,\ \ldots\ \rangle
\end{aligned}$$

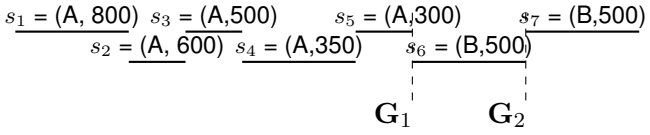


Figure 3.5: The vector of gaps, $\mathbf{G}$

Using this information, the error of merging the tuples $\{s_2, s_3\}$ into a tuple z is computed as $SSE(\{s_2, s_3\}, \{z\}) = 1\,890\,000 - 1\,280\,000 - \frac{(2\,700-1\,600)^2}{4-2} = 5\,000$.

3.5.3 Pruning the Search Space of the DP Scheme

Recall that filling the error matrix $\mathbf{E}$ involves computing the value of each cell (k, i) for all $k = 1, \ldots, c$ and $i = 1, \ldots, n$ using the above dynamic programming equation. That leads to an algorithm whose performance depends quadratically on the input size and linearly on the output size. In this section we introduce bounds for the variables i and j used in the equation this way improving the performance of the algorithm. Bounding the variable i allows us to avoid computing some $\mathbf{E}_{k,i}$ if that would in any way evaluate to infinity. Otherwise, we speed up the evaluation of $\mathbf{E}_{k,i} = \min_{k-1 \leq j < i}\{\mathbf{E}_{k-1,j} + SSE(\mathbf{s}_i \backslash \mathbf{s}_j, \rho(\mathbf{s}_i \backslash \mathbf{s}_j, 1))\}$ by reducing the value range of the variable j. The bounds depend on the positions of the non-adjacent tuple pairs in the sorted input relation $\mathbf{s}$. Therefore, the resulting performance improvements are data-dependent.

Let $\mathbf{G}$ be a vector that stores the positions of the non-adjacent tuple pairs in the sorted input relation, $\mathbf{s}$, i.e., $\mathbf{G}_m = l$ if $s_l, s_{l+1} \in \mathbf{s}$, $s_l \not\prec s_{l+1}$, is the m-th pair of non-adjacent tuples. We use the information in $\mathbf{G}$ to compute the bounds of i and j variables.

Example 3.13. For the ITA result of the running example we have $\mathbf{G} = \langle 5, 6 \rangle$, which is illustrated in Figure 3.5. The first pair of non-adjacent tuples is $s_5 \not\prec s_6$. The second pair is $s_6 \not\prec s_7$.

First, we determine an upper bound, i_{max}, for the variable i under which $\mathbf{E}_{k,i}$ does not evaluate to infinity. Intuitively, if the number of non-adjacent tuple pairs in $\mathbf{s}_i$ is greater than k, then merging across gaps is unavoidable, and, we are sure that the error $\mathbf{E}_{k,i}$ is infinite. As long as $k \leq |\mathbf{G}|$, the value $\mathbf{G}_k$ tells us the position of the k-th non-adjacent tuple pair. Consequently, the subset $\mathbf{s}_i = \{s_1, \ldots, s_{\mathbf{G}_k}\} \subseteq \mathbf{s}$ has $k - 1$ non-adjacent tuples pair and is the maximal subset that can be reduced to size k. Therefore, $i_{max} = \mathbf{G}_k$ and for all $i > i_{max}$ we have $\mathbf{E}_{k,i} = \infty$. When $k > |\mathbf{G}|$, the rule may no longer be applied and we set i_{max} equal to the size of the input relation, $i_{max} = |\mathbf{s}|$. Observe, that the more non-adjacent tuple pairs are present in

the relation, the more advantageous is this upper bound to speed up the evaluation.

Example 3.14. Consider the computation of $\mathbf{E}_{1,i}$ using the vector $\mathbf{G} = \langle 5, 6\rangle$ shown in Figure 3.5. The value $\mathbf{G}_1 = 5$ indicates that at most the first five tuples, $\mathbf{s}_5 = \{s_1, \ldots, s_5\}$, can be merged into one tuple without crossing a gap and inducing an infinite error. Therefore, given $k = 1$, the upper bound for i is $i_{max} = \mathbf{G}_1 = 5$; for all $i > 5$ we have $\mathbf{E}_{1,i} = \infty$. Given $k = 2$ the upper bound for i is $i_{max} = \mathbf{G}_2 = 6$. For all greater values of k the rule does not apply and i cannot be upper-bounded.

Second, whenever $E_{k,i}$ must be evaluated, we can determine a lower bound, j_{min}, for the variable j. The recursion formula for $\mathbf{E}_{k,i}$ determines the error of merging the tuples $\mathbf{s}_i \setminus \mathbf{s}_j$ into one tuple, which will be infinite if $\mathbf{s}_i \setminus \mathbf{s}_j$ contains at least one non-adjacent tuple pair. This is the case if j is smaller than the position of the right-most non-adjacent tuple pair in $\mathbf{s}_i$, if such a pair exists. The lower bound for j is therefore the position of the right-most non-adjacent tuple pair, i.e., $j_{min} = \max\{\mathbf{G}_l \mid \mathbf{G}_l < i \wedge l = 1, \ldots, |\mathbf{G}|\}$. If $\mathbf{s}_i$ contains no gaps, we set $j_{min} = k - 1$. Hence, to evaluate $\mathbf{E}_{k,i}$ it is enough to loop only over $j_{min} \leq j < i$. To efficiently determine j_{min}, we use binary search over $\mathbf{G}$. If there are no gaps in $\mathbf{s}_i$, the search does not return any result and we set $j_{min} = k - 1$. Interestingly, when $j_{min} = G_{k-1}$, then the subset $\mathbf{s}_i$ has exactly k gaps and the only choice to split $\mathbf{s}_i$ is at $j = j_{min}$.

Example 3.15. To compute $\mathbf{E}_{3,6}$, the basic DP scheme evaluates the SSE of merging the tuples $\mathbf{s}_6 \setminus \mathbf{s}_j$ for $j = 2, \ldots, 5$. The right-most non-adjacent pair in $\mathbf{s}_6$ is $j_{min} = \mathbf{G}_1 = 5$. Therefore, only for $j = 5$ the error is different from ∞; the error computation for $j = 2, \ldots, 4$ can safely be pruned.

3.5.4 The Size-Bounded PTA Algorithm

Algorithm 3.1 shows algorithm $\mathrm{PTA_c}$ for the evaluation of size-bounded PTA queries using the above DP scheme. First, the ITA result, $\mathbf{s}$, over the input relation $\mathbf{r}$ is computed using any ITA algorithm. (We assume $\mathbf{s}$ to be sorted by the grouping attributes $\mathbf{A}$ and, within each group, in chronological order; if not, an additional sorting step is required.) Next, the vectors $\mathbf{G}$, $\mathbf{L}$ and matrices $\mathbf{S}$, $\mathbf{SS}$ that are needed for the error computation are initialized. Notice that this initialization could be pushed into the ITA algorithm to avoid an additional scan of $\mathbf{s}$. Next, the error, $\mathbf{E}$, and split point, $\mathbf{J}$, matrices are initialized. The code in the following for loop fills them up implementing the DP scheme and performance improvements described before. For each matrix row, k, we iterate over columns, i, computing $\mathbf{E}_{k,i}$. The upper bound for i is obtained from the gap vector $\mathbf{G}$. When $k = 1$ we implement the first condition in the scheme and evaluation of $\mathbf{E}_{k,i}$ is straightforward. The

following lines implement the second condition. When the number of non-adjacent pairs in the subset $\mathbf{s}_i$ equals to k, then the only possible split point is at $j = j_{min}$. In other cases, iteration over j is necessary. We lower-bound the variable j, that is, j must be greater than the index of the right-most non-adjacent pair in the subset $\mathbf{s}_i$. Recall that $\mathbf{E}_{k,i}$ has been initialized to infinity. Then, iterating over j in decreasing order, we choose any smaller value. It has been shown in [27] that j should be iterated in decreasing order, i.e. from $i-1$ towards j_{min}. Since the value of err_2 increases with each iteration, the loop can be safely broken when e_2 alone exceeds the smallest error $\mathbf{E}_{k,i}$ found so far. Finally, the final while loop computes the output using the split point matrix $\mathbf{J}$ as described before.

Example 3.16. The evaluation of $\mathrm{PTA_c}$ over the **proj** relation starts with the computation of the ITA result s. The tuples are enumerated from 1 to 7 as in Figure 3.1(c). Next, $\mathbf{E}_{1,i}$ is computed for all $i = 1, \ldots, 5$. The values $\mathbf{E}_{1,6}$ and $\mathbf{E}_{1,7}$ are infinite and their evaluation will be avoided with the help of upper bound i_{max}. Similarly, we compute $\mathbf{E}_{2,i}$ for all $i = 2, \ldots, 6$ and avoid evaluation of $\mathbf{E}_{7,2}$. When $k = 2$ and i is between 2 and 5, the loop over j ranges between 1 and i. However, when i is 6, the value of j is fixed at 5. This way all the remaining errors are computed until $k = 4$ and $i = 7$, and the final output relation shown in Figure 3.1(d) is produced.

The runtime complexity of $\mathrm{PTA_c}$ depends on the ITA algorithm and the merging step. We assume that ITA is computed by one of several algorithms that have been proposed in the past, e.g. [12, 34, 43]. Their average running time is $O(n \log n)$, where n is the size of the input relation. In the merging step we evaluate the error within three nested loops, one per variable k, i, and j. The first two perform at most c and n iterations, respectively. The maximum number of iterations in j loop is equal to the size of the largest adjacent tuple subset in the ITA result, q. Error evaluation takes $O(p)$ time for p aggregation functions, however, p is usually insignificantly small and can be regarded as a constant. Therefore, the running time complexity of the merging step in the $\mathrm{PTA_c}$ algorithm is $O(c\,n\,q)$. In the worst case, when the dataset has no temporal gaps or aggregation groups, $q = n$ and the complexity of $\mathrm{PTA_c}$ is $O(n^2\,c)$. The space complexity of the algorithm is $O(n^2)$ as the split point matrix, $\mathbf{J}$, must be kept in memory entirely. On the other hand, only the two most recent rows of the error matrix, $\mathbf{E}$, are necessary.

3.5.5 The Error-Bounded PTA Algorithm

To answer error-bounded PTA queries, we use the same DP scheme as for the size-bounded algorithm. The DP solution computes all the optimal reductions to $k = 1, 2, \ldots$ tuples in increasing order of k. As k increases,

Algorithm: $\mathrm{PTA_c}(\mathbf{r}, \mathbf{A}, \mathbf{F}, c)$
$\mathbf{s} \leftarrow \mathcal{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{r}$;
Initialize $\mathbf{G}, \mathbf{L}, \mathbf{S}, \mathbf{SS}$;
Initialize $\mathbf{E}, \mathbf{J}$ to ∞ and 0, respectively;
for $k = 1, \ldots, c$ **do**
  if $k \leq |\mathbf{G}|$ **then** $i_{max} = \mathbf{G}_k$ **else** $i_{max} = |\mathbf{s}|$;
  for $i = k, \ldots, i_{max}$ **do**
    if $k = 1$ **then**
      $\mathbf{E}_{1,i} \leftarrow SSE(\mathbf{s}_i, \rho(\mathbf{s}_i, 1))$;
      $\mathbf{J}_{1,i} \leftarrow 0$;
    else
      $j_{min} \leftarrow \max\{k-1, \mathbf{G}_l | \mathbf{G}_l < i \wedge l = 1, \ldots, |\mathbf{G}|\}$;
      if $\mathbf{G}_{k-1} = j_{min}$ **then**
        $j \leftarrow j_{min}$;
        $\mathbf{E}_{k,i} \leftarrow \mathbf{E}_{k-1,j} + SSE(\mathbf{s}_i \setminus \mathbf{s}_j, \rho(\mathbf{s}_i \setminus \mathbf{s}_j, 1))$;
        $\mathbf{J}_{k,i} \leftarrow j$;
      else
        for $j = i-1, \ldots, j_{min}$ **do**
          $err_1 \leftarrow \mathbf{E}_{k-1,j}$;
          $err_2 \leftarrow SSE(\mathbf{s}_i \setminus \mathbf{s}_j, \rho(\mathbf{s}_i \setminus \mathbf{s}_j, 1))$;
          if $err_1 + err_2 < \mathbf{E}_{k,i}$ **then**
            $\mathbf{E}_{k,i} \leftarrow err_1 + err_2$;
            $\mathbf{J}_{k,i} \leftarrow j$;
          if $err_2 > \mathbf{E}_{k,i}$ **then break**;
$\mathbf{z} \leftarrow \emptyset$, $n \leftarrow |\mathbf{s}|$;
while $c > 0$ **do**
  $j \leftarrow \mathbf{J}_{c,n}$;
  $\mathbf{z} \leftarrow \mathbf{z} \cup \{s_{j+1} \oplus \ldots \oplus s_n\}$;
  $n \leftarrow j$; $c \leftarrow c - 1$;
return $\mathbf{z}$;

Algorithm 3.1: The $\mathrm{PTA_c}$ algorithm for size-bounded PTA

1 **Algorithm:** $\mathrm{PTA}_\epsilon(\mathbf{r}, \mathbf{A}, \mathbf{F}, \epsilon)$
2 $\mathbf{s} \leftarrow \mathcal{G}^{ITA}[\mathbf{A}, \mathbf{F}]\mathbf{r}$;
3 $\mathbf{E}_{max} \leftarrow SSE(\mathbf{s}, \rho(\mathbf{s}, c_{min}))$;
4 Initialize $\mathbf{S}, \mathbf{SS}, \mathbf{L}, \mathbf{G}, \mathbf{E}, \mathbf{J}$;
5 **for** $k = 1, \ldots, |\mathbf{s}|$ **do**
6 Fill $\mathbf{E}, \mathbf{J}$ using lines 6–24 in Algorithm 3.1;
7 **if** $\mathbf{E}[|\mathbf{s}|, k] \leq \epsilon \cdot \mathbf{E}_{max}$ **then**
8 $c \leftarrow k$;
9 **break**;
10 Build output $\mathbf{z}$ using lines 25–29 in Algorithm 3.1;
11 **return** $\mathbf{z}$;

Algorithm 3.2: The PTA_ϵ algorithm for the error-bounded PTA

the error monotonically decreases. Thus, the relation with the smallest k that satisfies the error bound ϵ is the solution.

Algorithm 3.2 depicts the evaluation algorithm for error-bounded PTA, which is very similar to the algorithm in Algorithm 3.1. The variable E_{max} is set to the maximum non-infinite error, i.e., the error that would be induced by merging all adjacent tuples together. This value can be computed together with ITA result at no additional cost. Then, in the main loop k iterates from 1 to the cardinality of the ITA relation, where the error matrix, $\mathbf{E}$, and the split point matrix, $\mathbf{J}$, are computed. The loop is terminated when the error exceeds ϵ. The running time complexity of this algorithm is the same as of PTA_c.

3.6 Experimental Evaluation

In this section we evaluate experimentally the reduction capability of PTA to reduce the ITA result. To answer the question we use both synthetic and real-world datasets. We measure the error that the dynamic programming algorithm induces at various reduction ratios. Intuitively, the operator performs well when it introduces low errors at high reduction ratios. Having evaluated the capabilities we turn to measure the scalability of the dynamic programming algorithm.

3.6.1 Experimental Setup

We implemented the algorithms using Java$^{\mathrm{TM}}$ version 6. They are executed on a Linux machine with 4 AMD 2600MHz Opteron processors and 16GB of RAM. An Oracle 11g database running on the same machine is used as the data storage medium.

Table 3.1: The aggregation queries used for evaluation of PTA

(a) ETDS

Name	Grouping	Agg. Functions	ITA Size	c_{min}
E1	–	avg	6 394	1
E2	–	max	6 394	1
E3	–	sum	6 394	1
E4	Emp.No., Dep.	avg	5 419 493	339 067

(b) Incumbents

Name	Grouping	Agg. Functions	ITA Size	c_{min}
I1	Dep., Proj.	avg	16 144	131
I2	Dep., Proj.	max	16 144	131
I3	Dep., Proj.	sum	16 144	131

(c) Time series

Name		Dimensionality	ITA Size	c_{min}
T1	Chaotic	1	1 800	1
T2	Tide	1	8 746	1
T3	Wind	12	6 574	216

(d) Synthetic

Name	Grouping	Dimensionality	ITA Size	c_{min}
S1	–	10	10 000 000	1
S2	yes	10	10 000 000	50 000

For the experiments we use the following data relations: real-world Incumbents dataset kindly donated by the University of Arizona, USA; synthetic employee temporal dataset (ETDS) donated by F. Wang [57]; a subset of real-world time series data from the UCR Time Series Data Repository [30]; a large synthetic dataset for large scale experiments. Our aim is to evaluate the behavior of PTA in the context of various aggregation functions, the presence and absence of grouping attributes, and varying numbers of temporal gaps. Therefore, as summarized in Table 3.1, we issue different aggregation queries over the four temporal relations. The resulting datasets sport different attribute value distribution, as well as different number of aggregation groups and temporal gaps. Next we discuss the properties of each of the datasets.

The ETDS relation depicts the evolution of employees in a company and contains 2 875 697 records. Each record stores employee number, sex, department, title, salary, and contract validity interval in months. The ITA queries over this relation are summarized in Table 3.1(a). The queries E1, E2, and E3 specify different aggregation functions but no grouping. They yield 6 394 tuple ITA result relations. These relations have no temporal gaps nor aggregation groups and thus their c_{min} is 1. Observe that the results

of these queries may have very different attribute values as they have been obtained using different aggregation functions. The query E4 specifies the grouping by employee number and department and the corresponding ITA result yields more than five million tuples. This query illustrates the case when the size of the ITA result may exceed the input size.

The Incumbents relation records the change of employee salaries over time. With each tuple a project ID, department ID, salary, and time interval in months are associated. The relation has 83 857 records. The corresponding ITA queries, summarized in Table 3.1(b), use project ID and department ID as grouping attributes and yield 16 144 tuples.

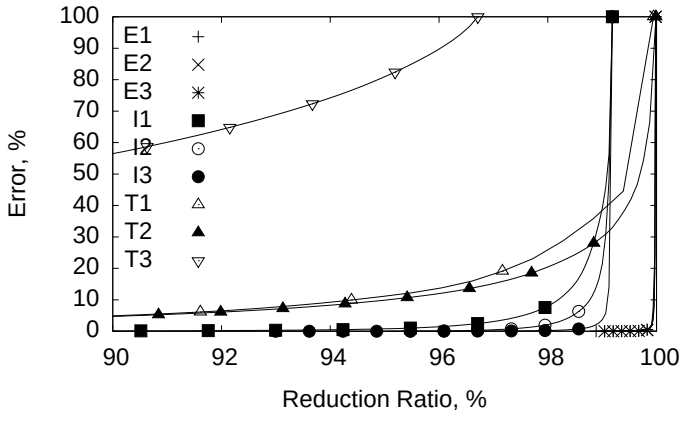
Time series datasets from UCR Time Series Data Repository [30] offer a great variety of data from various real-world sources. Each time series record has one or more aggregate values and a timestamp that we extend into a validity interval of length 1 obtaining sequential relations. Thus, time series data can be passed directly to PTA merging step omitting ITA aggregation. Table 3.1(c) summarizes the time series we use in this work. T1, Chaotic, as well as T2, Tide, has one aggregate value, 1 800 and 8 746 samples respectively. The T3, Wind, series has 12 aggregate values (dimensions).

Finally, to avoid any data induced bias we generate a synthetic dataset with ten million tuples, one grouping attribute and ten aggregate attributes whose values are distributed uniformly. Table 3.1(d) summarizes the two ITA queries we issued over this relation. The query S1 does not specify grouping. The result is a sequential relation with no temporal gaps, i.e. $c_{min} = 1$. The query S2, on the other hand, uses grouping. The result has 50000 groups with 200 sequential tuples in each. We use these datasets and their subsets to evaluate the scalability of PTA algorithms as well as the influence of varying dimensionality on the quality of PTA results.

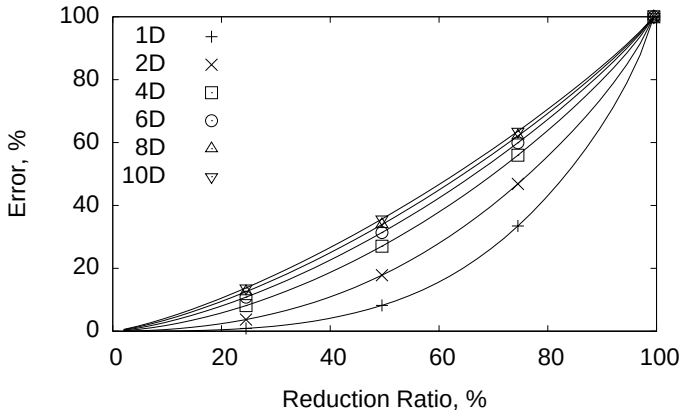
3.6.2 PTA Reduction Capability

In the following set of experiments we evaluate PTA algorithms by quantifying the error they induce when reducing the ITA result to different sizes. Ideally, the error should be low when the reduction ratio is high, indicating that the operator effectively discards redundant information borne by the data.

We quantify the error induced by the $\mathrm{PTA_c}$ algorithm at every possible output size. Note that PTA_ϵ results are identical and we do not report them here. We use every dataset except E4 and the synthetic one as these two are too large to be processed in a reasonable time. The error and output size values were normalized to fit in the range between 0 and 100%. Thus we obtained error growth curves where the horizontal axis depicts reduction ratio and the vertical one corresponds to the error. In particular, any ITA



(a)



(b) Synthetic Data

Figure 3.6: PTA error as a function of reduction ratio over (a) different datasets; (b) varying dimensionality

result has 0% reduction and thus 0% error; 100% error is reached when the dataset is reduced to its c_{min}.

Figure 3.6(a) depicts the error curves in the range of 90% to 100% reduction. Observe that for most queries the error remains very low even at huge reduction ratios. That is especially true for queries over the ETDS data. Queries over the Incumbents relation as well as the time series datasets can be reduced significantly as well. For example, the series T1 is reduced by 95%, i.e. 100 tuples, before the total error exceeds 10%. Only the T3 series reaches 55% error by 90% of reduction.

The relatively bigger PTA errors on the only high dimensional dataset, T3, suggest that the reduction capabilities depend on the dimensionality of the data. We evaluate multiple PTA queries over a sequential 2 000 tuple

subset of the synthetic dataset using $\mathrm{PTA_c}$. For each query we specify different number of aggregation attributes. Figure 3.6(b) depicts the resulting error growth curves. Observe that as dimensionality increases the overall error grows as well.

The above results allow us to conclude that in most cases the PTA operator helps to reduce the result size significantly and only induce small errors. The reduction capabilities do not depend on the aggregation functions or grouping attributes specified in the query, but on the data dimensionality. That is not surprising as the dimensionality related problems have long been known [11].

3.6.3 Scalability Evaluation

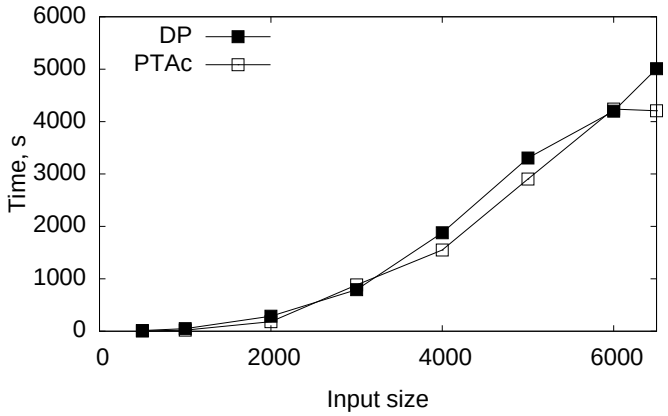
We evaluate running time performance of the dynamic programming approach. In the following performance graphs we measure the total CPU time devoted for query evaluation. We exclude the time taken to produce ITA results as well as the time needed to write the final PTA result back to the database.

We take as our baseline the straight-forward implementation of the dynamic programming equation (DP) as described in Sec. 3.5.1. We compare the time that DP takes to evaluate PTA queries to that of the $\mathrm{PTA_c}$ algorithm that implements the improvements developed in this paper as described in Sec. 3.5.4. We expect the $\mathrm{PTA_c}$ algorithm to be faster when dealing with data that sports multiple temporal gaps and aggregation groups and otherwise similar to that of DP. In these experiments there is no difference between $\mathrm{PTA_c}$ and $\mathrm{PTA_\epsilon}$, therefore we use the former.

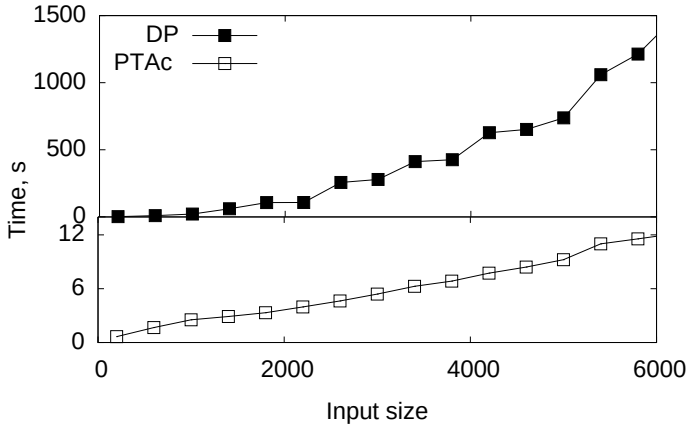
Figures 3.7(a) and 3.7(b) illustrate the influence of the input size to the computation time. In the first experiment, Figure 3.7(a), we use varying size sequential subsets of the synthetic data, i.e. they have no temporal gaps or aggregation groups. We vary the size of the input from 500 tuples to 6 500 whereas the output, c, and dimensionality, p, are fixed at 500 and 10 respectively. As expected, the two approaches show no significant difference.

Next, in Figure 3.7(b), we observe that the performance of $\mathrm{PTA_c}$ improves significantly when the argument relation contains multiple groups or gaps. In this instance we used subsets of grouped synthetic data. For each different size input we keep the number of aggregation groups fixed at 200 increasing only the number of tuples within each group. As the figure depicts, the $\mathrm{PTA_c}$ algorithm significantly outperforms DP and scales almost linearly as the presence of gaps reduces the amount of computation.

In Figure 3.8 we see how the change of the output size, c, influences the performance. Here we use 2 000 tuples from the synthetic dataset with 200 groups and ten tuples in each group. The value of c varies from 1 to 2 000 and n, p remain fixed. As expected, the running time increases linearly with



(a) No gaps



(b) With gaps

Figure 3.7: $\mathrm{PTA_c}$ running time as a function of input size

increasing size of the output. Observe however, that the $\mathrm{PTA_c}$ algorithm is not overly sensitive to the size bound, c, as the presence of gaps is the most important speed factor.

To conclude, the experiments confirm the estimated performance of the $\mathrm{PTA_c}$ algorithm. Applying it in realistic situations when data has temporal gaps, or multiple aggregation groups are specified in the query yields the result much faster than the plain dynamic programming approach does.

3.7 Summary

In this chapter we defined the parsimonious temporal aggregation operator, PTA, for size- and error-bounded queries. By reducing the instant temporal

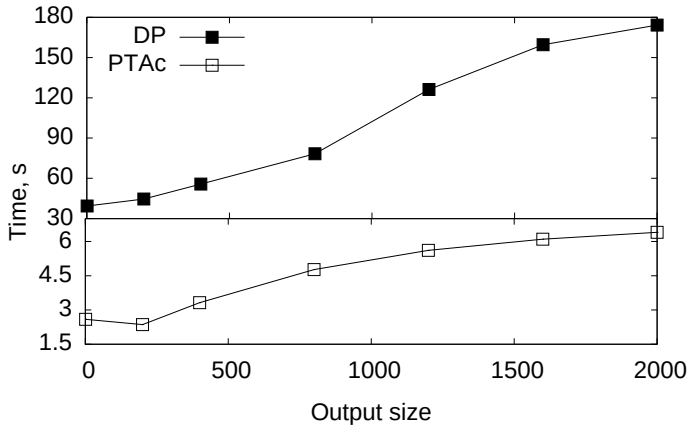


Figure 3.8: $\mathrm{PTA_c}$ running time as a function of output size, data with gaps

aggregation result via approximation, the approach allows us to control the size of the aggregation result thus overcoming limitations of existing operators. Since PTA is an approximating operator, its result may occasionally seem unexpected at the first glance. For example, when using avg as the sole aggregation function, PTA will merge two adjacent tuples that have very similar average values although the underlying data that contributes to these average values may be vastly different. In such a situation the user may instruct PTA to compare underlying distributions by specifying additional aggregation functions commonly used in descriptive statistics, such as standard deviation. Consequently, PTA will merge only those adjacent tuples that have similar average and standard deviation.

Another possible drawback of the operator is its inability to cope with noise in the data. While evaluating the operator we have not come up with such data, although it is easy to imagine situations when measurements obtained from sensors are not reliable or news articles harvested from the Internet are accompanied with large volumes of spam. For example, consider adding one tuple with very high Sal value anywhere in the middle of our **proj** relation. That would separate otherwise very similar ITA tuples making it impossible for PTA to merge them without induction of very high error. In fact, such noise in temporal data is largely ignored in current research literature and the users have to cleanse the data before they apply temporal data analysis tools. While evaluating this thesis we have worked with noiseless data only, such as employment contracts. In the future work we will address the issue of noise using methods similar to those employed in nonparametric spline smoothing techniques [17].

Applications requiring the most precise aggregation result should evaluate PTA queries using suggested dynamic programming algorithms, $\mathrm{PTA_c}$ and PTA_ϵ, that introduce minimal approximation error. In realistic situa-

tions, when data contains temporal gaps, or aggregation groups are specified in the query, the algorithms scale linearly with respect to the size of the input. In the worst case, they will take $O(n^2c)$ time and $O(n^2)$ space to find the reduction of n tuples in the input relation to c tuples. Experimental evaluation proved that the PTA operator can reduce the ITA result significantly yet introduce only small errors.

CHAPTER 4

Greedy Evaluation of PTA

In this chapter we continue elaborating on the parsimonious temporal aggregation operator (PTA). We focus on the online evaluation of PTA queries where algorithmic performance is of the utmost importance. We propose an efficient greedy merging strategy with a precision that is upper bounded by $O(\log n)$. We present two algorithms that implement this strategy and begin to merge as ITA tuples are produced. They require $O(n \log(c + \beta))$ time and $O(c + \beta)$ space, where β is the size of a read-ahead buffer and is typically very small. An empirical evaluation on real-world and synthetic data shows that PTA considerably reduces the size of the aggregation result, yet introducing only small errors. The greedy algorithms are scalable for large datasets and introduce less error than other approximation techniques.

4.1 Introduction

Parsimonious temporal aggregation queries are likely to be issued over huge argument relations storing archives of historical data. The dynamic programming approach can be used to evaluate them offline introducing minimal possible error. However, many applications would benefit from a less exact answer available immediately, i.e. computed in an online fashion. For example, a data analyst may want to obtain an approximate answer to her PTA query before she tells the system to start the computation of the optimal reduction.

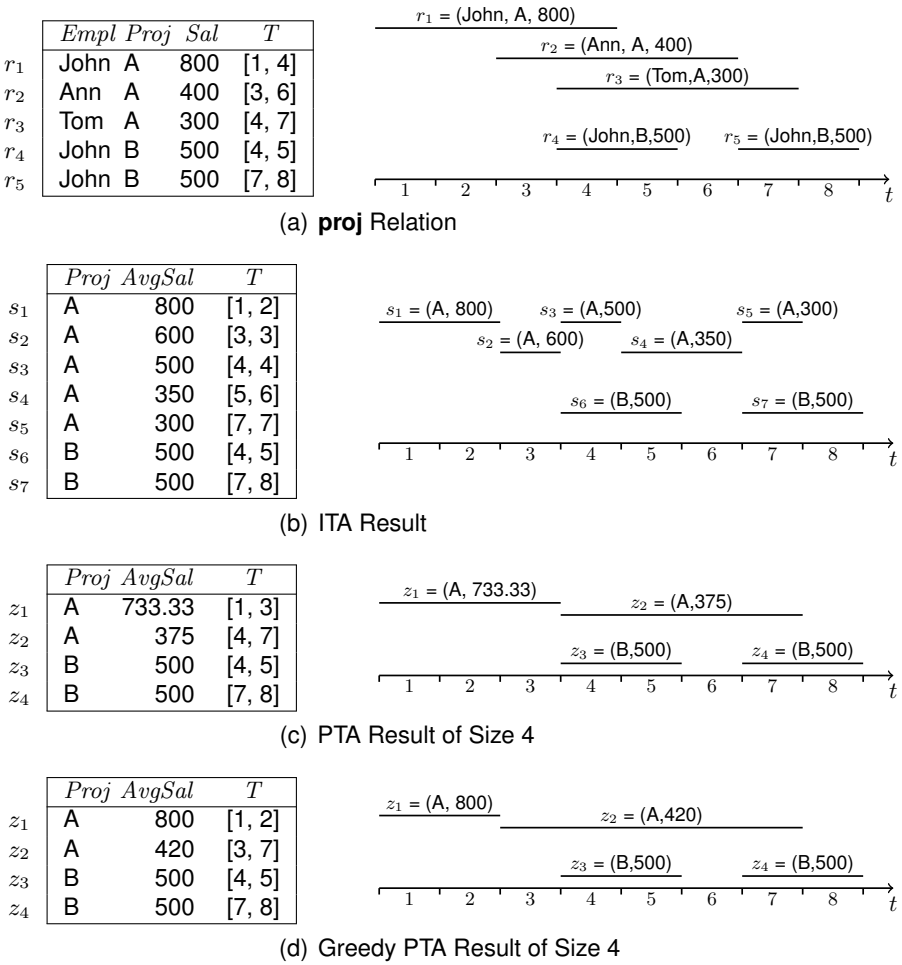
In this chapter we present a greedy evaluation strategy for PTA queries. We propose to merge greedily and iteratively pairs of the most similar tuples in ITA result until the size- or error-bound is satisfied. Such a strategy may not provide the optimal result, however, we prove that the additional error it induces is reasonably small and upper-bounded.

Furthermore, we show that greedy merging can commence before the whole ITA relation is computed and that it would not incur any additional error. We introduce two novel algorithms for the greedy evaluation of size- and error-bounded PTA queries, $\mathrm{gPTA_c}$ and $\mathrm{gPTA_\epsilon}$ respectively. The algorithms read ITA tuples as they arrive and attempt merging as soon as possible. They maintain in memory only the intermediate PTA result that takes $O(c+\beta)$ space, where β is typically much smaller than the whole ITA relation. Reducing computation space requirements results in better running time performance and the algorithms need only $O(n\log(c+\beta))$ time to compute the reduction of size c from n ITA tuples.

Finally, we conduct extensive experimental evaluation using real-world as well as synthetic data. We show that the greedy algorithms scale well for large datasets. We compare the greedy algorithms to other plausible solutions suggested in the field of time series approximation. The experiments reveal that the greedy strategy introduces significantly less error than other state of the art methods.

We build on top of the concepts developed in Chapter 3. Therefore, we continue using the same notation and adhere to the same database model as defined previously. We start our discussion by introducing a small example dataset that we use to illustrate the concepts. Next, we introduce the greedy merging strategy and show that its definition requires a notion of tuple similarity. We derive the similarity measure from the sum square error of the previous chapter. We also prove the upper bounds of the error that the greedy approach may introduce. We continue by introducing greedy evaluation algorithms that implement the strategy in an efficient way. Note that the algorithms are different for size- and error-bounded PTA queries.

4.2 Example



	Empl	*Proj*	*Sal*	T
r_1	John	A	800	[1, 4]
r_2	Ann	A	400	[3, 6]
r_3	Tom	A	300	[4, 7]
r_4	John	B	500	[4, 5]
r_5	John	B	500	[7, 8]

(a) **proj** Relation

	Proj	*AvgSal*	T
s_1	A	800	[1, 2]
s_2	A	600	[3, 3]
s_3	A	500	[4, 4]
s_4	A	350	[5, 6]
s_5	A	300	[7, 7]
s_6	B	500	[4, 5]
s_7	B	500	[7, 8]

(b) ITA Result

	Proj	*AvgSal*	T
z_1	A	733.33	[1, 3]
z_2	A	375	[4, 7]
z_3	B	500	[4, 5]
z_4	B	500	[7, 8]

(c) PTA Result of Size 4

	Proj	*AvgSal*	T
z_1	A	800	[1, 2]
z_2	A	420	[3, 7]
z_3	B	500	[4, 5]
z_4	B	500	[7, 8]

(d) Greedy PTA Result of Size 4

Figure 4.1: The **proj** relation and greedy PTA evaluation of the query *"What is the average monthly salary for each project?"*

Throughout this chapter we use the **proj** relation that is also used in Chapter 3 and shown in Figure 4.1(a). Figure 4.1(b) shows the result of an ITA query *"What is the average monthly salary for each project?"*. Figure 4.1(c) shows the result of a PTA query *"What is the average monthly salary for each project represented using 4 tuples?"*. The result is obtained by applying the dynamic programming strategy introduced in Chapter 3. It offers an optimal reduction of the ITA relation, i.e. the introduced error is minimal.

Figure 4.1(d), on the other hand, depicts the result obtained by greedily reducing the ITA result. At each iterative step we choose a pair of tuples that introduce the smallest additional error and merge them. We continue in

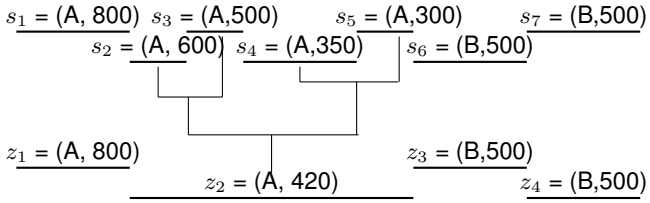


Figure 4.2: The dendrogram of the greedy merging steps.

the same fashion until the size bound is satisfied. Observe that the resulting relation is slightly different from that obtained using the dynamic programming strategy. Nevertheless it provides a good approximation of the ITA result.

4.3 Greedy Merging Strategy

Let s be an ITA result relation. The *greedy merging strategy* (GMS) reduces s in an iterative manner until the size or error bound is satisfied. At each step, it operates on an intermediate result relation, choosing from it a pair of the most similar tuples for merging. Intuitively, the less the error of merging two tuples is, the more similar they are. By replacing the chosen pair with the newly merged tuple a reduced intermediate result is obtained. When several pairs of tuples are equally similar, any pair can be chosen. We elect to merge the pair with the smallest timestamp value. This choice, however, does not influence the total error induced by the greedy merging process.

Example 4.1. Figure 4.2 illustrates the greedy merging steps over the example ITA relation with size bound $c = 4$. The first tuples to be merged (i.e., the most similar ones) are s_4 and s_5 followed by s_2 and s_3. The two new tuples are then merged to produce the final result tuple z_2. The result of greedy merging differs from the precise PTA result in Figure 4.1(c), where tuple s_2 is merged with s_1. The DP algorithm induces an error of 49 166, while the error of the greedy approach is 63 000, yielding an error ratio of 1.28 between the two.

In order to apply GMS, the notion of similarity between tuple pairs needs to be precisely defined. Consider a sequential relation s' that is obtained from the initial relation, s, by applying the reduction operator. Merging a pair of adjacent tuples $s_i \in \mathrm{s}', s_j \in \mathrm{s}'$ leads to a new relation, say $\mathbf{z}$. Then, the *dissimilarity* of the tuples s_i and s_j is the error introduced by the merge, i.e., $dsim(s_i, s_j) = SSE(\mathbf{s}, \mathbf{z}) - SSE(\mathbf{s}, \mathbf{s}')$. In order to determine the dissimilarity using this equation, the source relation s must be available, which is not practical if we want to start merging before the whole ITA result is computed. The following proposition shows that $dsim(s_i, s_j)$ can be determined

by considering only the tuples s_i and s_j.

PROPOSITION 4.1. *Let* s *be a sequential relation,* $\mathrm{s}' = \rho(\mathrm{s}, k)$ *be a reduction to size* k*, and* $\mathbf{z}$ *be obtained from* s' *by merging the tuples* $s_i \in \mathrm{s}', s_j \in \mathrm{s}'$ *to* $s_i \oplus s_j = z \in \mathbf{z}$*. The dissimilarity of the tuples* s_i, s_j *is* $dsim(s_i, s_j) = SSE(\{s_i, s_j\}, \{z\})$*.*

Proof. Let $\mathbf{s}_i^* \subset \mathrm{s}$ be the tuples in s that make up s_i, i.e., for all $s \in \mathbf{s}_i^*$ we have $s[\mathbf{A}] = s_i[\mathbf{A}] \wedge s.T \subseteq s_i.T$. Let $\mathbf{s}_j^*$ be defined for s_j in a similar fashion. Then, $\mathbf{s}^* = \mathbf{s}_i^* \cup \mathbf{s}_j^*$ constitute z. Since the sets $\mathrm{s}' \setminus \{s_i, s_j\}$ and $\mathbf{z} \setminus \{z\}$ are identical we rewrite $dsim(s_i, s_j)$ as

$$\begin{aligned} dsim(s_i, s_j) &= SSE(\mathbf{s}^*, \{z\}) - SSE(\mathbf{s}^*, \{s_i, s_j\}) \\ &= SSE(\mathbf{s}_i^*, \{z\}) - SSE(\mathbf{s}_i^*, \{s_i\}) + \qquad (a) \\ &\quad SSE(\mathbf{s}_j^*, \{z\}) - SSE(\mathbf{s}_j^*, \{s_j\}) \qquad (b) \end{aligned}$$

The summation $(a) + (b)$ is possible because the tuples s_i, s_j are adjacent, i.e., they do not overlap. Recall, that according to the merge function, $s_i = \frac{1}{|s_i.T|} \sum_{s \in s_i^*} |s.T| s$. Using the definition of the error, we can rewrite equation (a) (and similarly (b)) as

$$\begin{aligned} (a) &= \sum_{s \in s_i^*} |s.T|(s - z)^2 - \sum_{s \in s_i^*} |s.T|(s - s_i)^2 \\ &= |s_i.T|(z^2 - 2z \cdot s_i + s_i^2) = SSE(\{z\}, \{s_i\}). \end{aligned}$$

Since the square sum error is defined as a sum of individual errors and s_i, s_j are adjacent, we have

$$\begin{aligned} dsim(s_i, s_j) = (a) + (b) &= SSE(\{z\}, \{s_i\}) + SSE(\{z\}, \{s_j\}) \\ &= SSE(\{z\}, \{s_i, s_j\}). \end{aligned}$$

□

In contrast to the DP approach, GMS does not necessarily compute an optimal ITA reduction. At every greedy merging step, there is a chance of making a sub-optimal decision. Therefore, the more merging steps are performed, the more additional error can be accumulated. The following theorem shows that the error ratio of the greedy and optimal solution is asymptotically upper-bounded by the logarithm of the number of merging steps. Experimentally we show that the greedy reduction is in fact very close to the optimal one.

THEOREM 4.1. *Let* s^n *be a sequential relation of size* n*. Let* s^c *be a reduction of* s^n *to* c *tuples obtained using the greedy merging strategy. Let* $\mathbf{z}$ *be an optimal reduction of* s^n *to* c *tuples obtained using* $\mathrm{PTA_c}$*. Then, the error ratio between the two solutions is*

$$\frac{SSE(\mathbf{s}^n, \mathbf{s}^c)}{SSE(\mathbf{s}^n, \mathbf{z})} \leq O(\log n).$$

Proof. By the definition of the greedy merging strategy, the error induced by the merge of two most similar tuples in $\mathbf{s}^k$ is

$$SSE(\mathbf{s}^k, \mathbf{s}^{k-1}) = \min_{\substack{s_i, s_j \in \mathbf{s}^k \\ s_i \prec s_j}} \{dsim(s_i, s_j)\}.$$

The minimum is upper-bounded by an average which is, in turn, upper-bounded by the total error, i.e.,

$$\begin{aligned} SSE(\mathbf{s}^k, \mathbf{s}^{k-1}) &\leq \frac{1}{k-1} \sum_{\substack{s_i, s_j \in \mathbf{s}^k \\ s_i \prec s_j}} dsim(s_i, s_j) \\ &\leq \frac{1}{k-1} C \cdot SSE(\mathbf{s}^n, \mathbf{z}). \end{aligned}$$

C is a constant whose minimal value may vary depending on k, yet there is a value that satisfies any k. According to Proposition 4.1, the error $SSE(\mathbf{s}^n, \mathbf{s}^c)$ made by the greedy algorithm is the sum of errors made at each intermediate merging step, i.e., $SSE(\mathbf{s}^n, \mathbf{s}^c) = \sum_{k=n}^{c+1} SSE(\mathbf{s}^k, \mathbf{s}^{k-1})$. Replacing the summand with the upper-bound we get $SSE(\mathbf{s}^n, \mathbf{s}^c) \leq C \cdot SSE(\mathbf{s}^n, \mathbf{z}) \sum_{k=n}^{c+1} \frac{1}{k-1}$, which leads to the error bound of the theorem, namely $\frac{SSE(\mathbf{s}^n, \mathbf{s}^c)}{SSE(\mathbf{s}^n, \mathbf{z})} \leq C \cdot \sum_{k=n}^{c+1} \frac{1}{k-1} \leq O(\log n)$. □

A straight-forward implementation of the greedy merging strategy is to use a priority queue (e.g., a binary heap) to find the most similar tuple pairs. After inserting all ITA tuple pairs in the heap, the merging process starts, taking $O(n \log n)$ time and $O(n)$ space to compute any reduction of n ITA tuples. In the following we describe a more efficient implementation of the greedy merging strategy for size- and error-bounded PTA queries, which starts the merging process before the complete ITA result is available.

4.4 A Greedy Algorithm for Size-Bounded PTA

4.4.1 Basic Idea

We present the $\mathrm{gPTA_c}$ algorithm for the evaluation of size-bounded PTA queries, which integrates the computation of the ITA result and greedy merging into one process. In a nutshell, $\mathrm{gPTA_c}$ reads ITA result tuples as they become available and inserts them into a binary heap, which allows to efficiently identify the most similar tuple pair for merging. Whenever the heap contains more than c tuples, the algorithm attempts to merge the tuple at the top with its immediate predecessor, which requires some care. The tuples are only merged if GMS operating on the whole ITA relation would also choose to merge them. Such situation can only be identified if

the last two tuples in the heap are non-adjacent as stated by the following proposition. At any time during the computation the heap contains at most $c + \beta$ tuples. Since β is typically small, $\mathrm{gPTA_c}$ improves over GMS in terms of running time and space efficiency.

Recall that the ITA result tuples come sorted along the aggregation groups, and, within each group, along the temporal dimension. We enumerate them from 1 to n, i.e., $\mathbf{s} = \{s_1, \ldots, s_n\}$. Given a subset of $\mathbf{s}$, the following proposition specifies when GMS will merge the most similar tuples in it.

PROPOSITION 4.2. *Let* $\mathbf{s}_{j+1} = \{s_1, \ldots, s_i, s_{i+1}, \ldots, s_j, s_{j+1}\}$ *be a part of the ITA result* $\mathbf{s} = \{s_1, \ldots, s_n\}$ *and* $s_i \prec s_{i+1}$ *be the most similar pair of tuples in* $\mathbf{s}_{j+1}$ *for some* $i \leq j$*. The greedy merging strategy operating on* $\mathbf{s}$ *merges* s_i *and* s_{i+1} *if* $j \geq c$ *and* $s_j \not\prec s_{j+1}$*.*

Proof. The GMS will not elect s_i, s_{i+1} for merging as long as there are more similar pairs in $\mathbf{s}$. Since s_i, s_{i+1} are most similar in $\mathbf{s}_{j+1}$, all pairs which are more similar than s_i, s_{i+1} (if any) must be in $\mathbf{s}_n \setminus \mathbf{s}_j$. Therefore, leaving s_i, s_{i+1} intact, the smallest intermediate relation produced by GMS is $\mathbf{s}' = \{s_1, \ldots, s_i, s_{i+1}, \ldots, s_j, s_{j+1} \oplus \cdots \oplus s_n\}$. Since $|\mathbf{s}'| \geq |\mathbf{s}_j| > c$, more merging steps are needed to reduce $\mathbf{s}'$ to size c. Since the two tuples $s_j, s_{j+1} \oplus \cdots \oplus s_n$ are not adjacent, the most similar pair in $\mathbf{s}'$ to be merged next can only be s_i, s_{i+1}. □

Example 4.2. Consider reading the first five tuples, $\mathbf{s}_5 = \{s_1, \ldots, s_5\}$, of the ITA result in Figure 4.2, and let the size bound for the PTA result be $c = 4$. The tuples $s_4 \prec s_5$ are the most similar ones, yet they cannot be merged since the tuple to be read next, s_6, might form an even more similar pair with s_5. Therefore, we read ahead an get $\mathbf{s}_6 = \{s_1, \ldots, s_6\}$. Since $s_5 \not\prec s_6$, the GMS has to make a merge in the first five tuples, independent of the tuples that will follow.

Proposition 4.2 provides a criterion to perform early merging, yet guaranteeing the same result as GMS: if more than c tuples are in the heap and the last tuple pair is not adjacent. However, when temporal gaps are rare or the aggregation groups are few, a large portion of the ITA result (in the worst case the whole result relation) may be inserted into the heap before a non-adjacent pair arrives. To avoid the heap size growing much beyond size c, we propose to heuristically determine whether the currently most similar tuple pair, s_i, s_{i+1}, would also be merged by GMS. Suppose that s_j is the last tuple in the heap and there is a more similar pair, s_k, s_{k+1}, in the ITA result that is connected to s_{i+1} by a sequence of adjacent tuples, but has not yet been inserted into the heap, i.e., $\mathbf{s}_{k+1} = \{s_1, \ldots, s_i, s_{i+1}, \ldots, s_j, \ldots, s_k, s_{k+1}\}$, where $s_{i+1} \prec \ldots \prec s_{k+1}$. GMS would first merge s_k, s_{k+1}. Since the merge result, $s_k \oplus s_{k+1}$, might

be more similar to s_{k-1} than s_i, s_{i+1} are, the new tuple is next merged with s_{k-1}. This might propagate back, and s_{i+1} may potentially become more similar to its new successor (which is the result of several merging steps) than to s_i. In such a situation, merging s_i, s_{i+1} would be a mistake, leading to a result that is likely to be different from the GMS result. However, the more tuples follow the current merge candidate, s_i, s_{i+1}, the lower is the probability that the similarity of s_i, s_{i+1} will be influenced by merging of subsequent tuples. Therefore, to keep the heap size small we use a parameter, δ, to specify the minimum number of adjacent tuples that have to follow the merge candidate for it to be merged. We will show experimentally that with $\delta = 1$ the difference between $\mathrm{gPTA_c}$ and GMS is negligible, yet space and performance gain is significant.

Observe that $c + \delta$ is essentially the lower bound of the heap size. In the worst yet unlikely case the heap size will be equal to the ITA result size. The higher is the value of δ parameter, the closer will be the final output to that of GMS. When $\delta = \infty$, $\mathrm{gPTA_c}$ and GMS produce the same output as shown by Theorem 4.2 below.

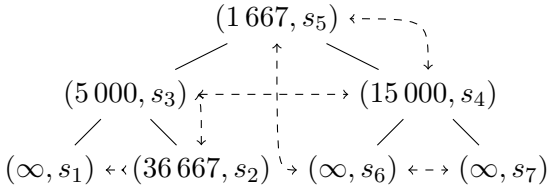
4.4.2 Heap Data Structure

We use a binary heap to avoid scanning the entire intermediate relation in search of the most similar pair of tuples. Given a relation s, we represent a tuple $s \in \mathrm{s}$ as a node N that records the following information: the sequence number of the tuple ($N.id$), the tuple itself (s), a pointer to the previous (in chronological order) node ($prev$), a pointer to the next node ($next$), and the error that would be induced by merging s with the previous tuple (key), i.e., $N.key = SSE(\mathrm{s}, \{N.s \oplus N.prev.s\})$. The key is set to ∞ if N and $N.prev$ represent non-adjacent tuples or s is the first tuple.

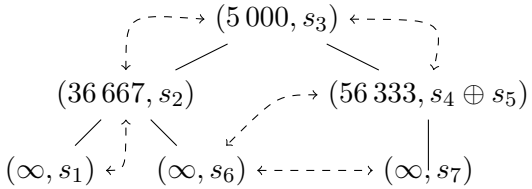
We define the following operations on the heap. INSERT creates a new node for a tuple and inserts it into the heap; this includes also the computation of the key value. PEEK returns the top node, N, but does not remove it from the heap. MERGE removes the top node, N, off the heap and merges the tuple $N.s$ into the preceding node, $P = N.prev$, yielding $P.s = P.s \oplus N.s$. The pointers $P.next$ and $N.next.prev$ are updated, the key values of $N.prev$ and of $N.next$ are recomputed, and the heap structure is updated. The field $P.id$ remains unchanged.

Example 4.3. Figure 4.3 depicts a binary heap. Solid lines represent parent-child relationships, dashed lines indicate $prev$ and $next$ links. In Figure 4.3(a) the heap contains the whole ITA result. The key of s_1 is infinite since s_1 is the first tuple, whereas the key of s_6 is infinite because s_5 and s_6 are not adjacent. The most similar tuple pair is s_4, s_5. Thus, the node on the peak represents s_5 with the key value 1 667, which is the error of merging s_4 and s_5. Figure 4.3(b) shows the heap after performing one

merge. The node s_5 is merged into the node s_4, which now contains $s_4 \oplus s_5$. The key value of $s_4 \oplus s_5$ and s_6 are re-evaluated, and the $next$ pointer of $s_4 \oplus s_5$ and the $prev$ pointer of s_6 are updated. The new peak node is s_3, thus s_2 and s_3 will be merged next.



(a) Before merging



(b) After merging

Figure 4.3: Binary heap used in the $\mathrm{gPTA_c}$ algorithm

4.4.3 $\mathrm{gPTA_c}$ Algorithm

The algorithm $\mathrm{gPTA_c}$ for the greedy evaluation of size-bounded PTA is shown in Algorithm 4.1. It takes the size-bound, c, and δ as input parameters and starts by initializing an empty heap, the ITA operator that produces sorted output, and three variables $LastGapId$, BG, and AG. The variable $LastGapId$ stores the sequence number of the last seen non-adjacent tuple pair, i.e. it is equal to the $N.id$ of the last seen node with infinite key value. The variables BG and AG store the number of tuples preceding and, respectively, succeeding the last non-adjacent node indicated by $LastGapId$. Trivial modifications to the ITA algorithm in [12] are necessary to allow processing the tuples one by one as they become available from the operator.

For each incoming ITA tuple a new node, N, is created and inserted into the heap. If $N.key$ is infinite, then the node represents a non-adjacent pair, the $LastGapId$ variable is set to the sequence number of this node and BG is increased by the value of AG. Otherwise, we increment the AG variable by 1.

Example 4.4. Consider having read the first 6 tuples of the ITA result in Figure 4.2. Since 5 tuples precede the latest gap, $BG = 5$. Only one tuple

follows the gap, thus $AG = 1$. Finally, the $LastGapId = 6$ variable indicates the non-adjacent tuple that follows immediately after the gap.

When the size of the heap exceeds the PTA size bound, c, the merging process starts with the second while loop. First a node with smallest key value is read from the heap. Then we check whether the merging can take place according to Proposition 4.2. If the condition evaluates to true, we proceed with merging. Otherwise, we make use of the heuristic and only check whether the node has at least δ adjacent successors. The higher is δ, the less likely it is that the choice to merge is different from that of GMS and that $\mathrm{gPTA_c}$ will deviate from GMS result. When $\delta = \infty$ the effect of the parameter is eliminated and merging will only happen when non-adjacent tuple pairs are discovered. In that case the algorithm is guaranteed to produce a result identical to that of GMS. If none of the two conditions evaluate to true, then merging cannot take place. We break the cycle and wait for more tuples to be inserted into the heap. Finally, when the whole ITA result is read and the heap still does not satisfy the size bound, we simply merge the remaining tuples.

Example 4.5. Consider running $\mathrm{gPTA_c}$ over the **proj** relation with $c = 3$ and $\delta = 1$. Figure 4.4 depicts each intermediate state of the heap. The ITA tuples come sorted: first the group "A" and only then the group "B". In (a) the heap contains the first four ITA tuples. The merging process can start as the most similar pair, s_2 and s_3, has 1 successor. Thus in (b) $s_2 \oplus s_3$ has been created to keep heap size equal to 3. Next, tuple s_5 is inserted leading to (c) where the tuple at the top of the heap is s_5. Even though the size of the heap exceeds c the merge cannot happen as at this point s_5 does not have δ successors. Observe that the tuple s_5 may happen to be more similar to s_6 (which is not known yet) and, thus, should be merged with it. When the tuple s_6 arrives, (d), it becomes clear that merging is possible. The heap now contains 5 tuples. The algorithm repeatedly merges adjacent tuples (e,f) until the size constraint is satisfied again. Specifically, s_4 is merged to s_5 and the result is then merged with $s_2 \oplus s_3$. Finally, with the arrival of tuple s_7 (g) the final result is computed as shown in (h). Over the whole process the heap contained at most 5 tuples whereas 7 ITA tuples have been processed.

THEOREM 4.2. *The output of the* $\mathrm{gPTA_c}$ *algorithm with* $\delta = \infty$ *is identical to that of GMS.*

Proof. The proof follows from Proposition 4.2. $\mathrm{gPTA_c}$ merges the same tuple pairs that GMS would merge. □

The complexity of $\mathrm{gPTA_c}$ depends on the ITA algorithm. Assume that the latter takes T time to produce a result relation of size n. In addition, it

```
1  Algorithm: gPTA_c(r, A, F, δ, c)
2  H ← new empty heap;
3  Initialize the ITA operator with F, A, and r;
4  LastGapId ← 0; BG ← 0; AG ← 0;
5  while s_i ← next tuple from G^ITA[A, F]r do
6      N ← INSERT(s_i);
7      if N.key = ∞ then
8          LastGapId ← N.id;
9          BG ← BG + AG;
10         AG ← 1;
11     else
12         AG ← AG + 1;
13     while |H| > c do
14         N ← PEEK();
15         if N.id < LastGapId ∧ BG ≥ c then
16             BG ← BG − 1;
17             MERGE();
18         else if N.id > LastGapId ∧ N has δ adjacent successors
           then
19             AG ← AG − 1;
20             MERGE();
21         else
22             break;
23 while |H| > c > c_min do
24     MERGE();
25 return H;
```

Algorithm 4.1: Greedy algorithm, $\mathrm{gPTA_c}$, for size-bounded PTA

uses S space for internal structures. Then, $\mathrm{gPTA_c}$ requires $O(T + n\log(c+\beta))$ time and $O(S + c + \beta)$ space (assuming that the heap operations take logarithmic time). In the worst case $c + \beta = n$, however for the majority of datasets we expect $c + \beta$ to be much smaller than n.

4.5 A Greedy Algorithm for Error-Bounded PTA

The algorithm, gPTA_ϵ, for the greedy evaluation of error-bounded PTA queries follows the same intuition as its size-bounded counterpart. As ITA result tuples are produced, it starts to merge tuples as early as possible and tries to merge as many tuples as possible before exceeding the error threshold ϵ. The major difference is on how to determine tuple pairs that

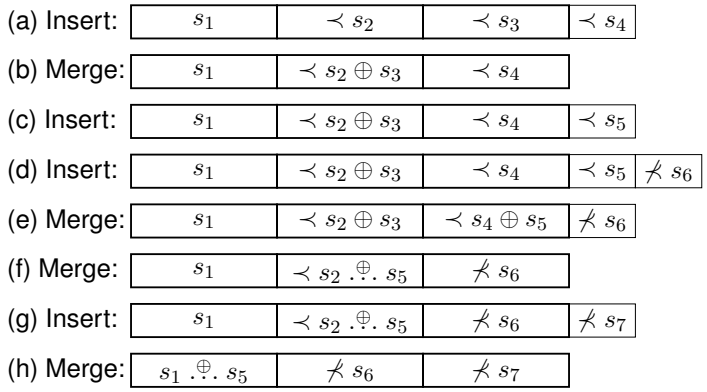


Figure 4.4: Reducing an ITA result to three tuples with $\mathrm{gPTA_c}$

would also be merged by GMS. For that we need to know the size, n, of the ITA result, s, and the maximal error, $E_{max} = SSE(\mathrm{s}, \rho(\mathrm{s}, c_{min}))$, of reducing s to smallest possible size, c_{min}. Since we do not wait until the ITA result is completed, these values have to be estimated. The ITA relation can be safely estimated to contain twice as many tuples as the argument relation. In order to estimate the total error we have to obtain a good sample of the ITA result. Using these two values, the following preposition shows how to identify tuple pairs in a subset of the ITA result that will be merged by GMS.

PROPOSITION 4.3. *Let $\mathrm{s}_{j+1} = \{s_1, \ldots, s_i, s_{i+1}, \ldots, s_j, s_{j+1}\}$ be a subset of the ITA result relation, $\mathrm{s} = \{s_1, \ldots, s_n\}$, $s_i \prec s_{i+1}$ be the most similar pair of tuples in s_{j+1} for some $i \leq j$, and $E_{max} = SSE(\mathrm{s}, \rho(\mathrm{s}, 1))$ be the maximum total error. The greedy merging strategy operating on s will merge s_i and s_{i+1} if $dsim(s_i, s_{i+1}) \leq \epsilon \frac{E_{max}}{n}$ and $s_j \not\prec s_{j+1}$.*

Proof. In the error-bounded setting GMS has to make as many merges as possible, yet introducing at most ϵE_{max} error. Let the total error introduced at some intermediate step be E_{tot}. GMS will elect s_i, s_{i+1} for merging at that step only if there is no other more similar pair among those in the intermediate result and $E_{tot} + \epsilon \frac{E_{max}}{n} \leq \epsilon E_{max}$. Since s_i, s_{i+1} are most similar in s_{j+1}, at most $n-j-1$ more similar tuple pairs may exist in $\mathrm{s}_n \backslash \mathrm{s}_j$. Once all of these tuples are merged, the total error will be $E_{tot} \leq (n-j-1) \cdot \epsilon \frac{E_{max}}{n}$. Since these merging steps do not change the similarity of s_i, s_{i+1} (due to $s_j \not\prec s_{j+1}$), s_i, s_{i+1} become now the most similar pair. Since $E_{tot} + \epsilon \frac{E_{max}}{n} \leq \epsilon E_{max}$, GMS will make at least one more merging step, choosing s_i, s_{i+1} for merging. □

To avoid the heap growing too much while we wait for a non-adjacent tuple pair, we adopt the same heuristic as for the size-bounded case. That is, the more tuples follow the current merge candidate, the more likely it is that this pair will be merged also by GMS. Hence, we use a user-specified

parameter δ to determine the minimum number of tuples that should follow the merge candidate for it to be merged.

Algorithm 4.2 shows the gPTA_ϵ algorithm for the greedy evaluation of error-bounded PTA. The variable E_{tot} is updated after each merging step and tracks the total error made so far. In addition, in line 6 we estimate the maximal total error, $\overline{E}_{max}$, and the size of ITA relation, $\overline{n}$. Incoming ITA tuples are inserted into the heap as they arrive and merging is attempted with each new tuple. Merging may only happen if the key of the node at the top of the heap is less than the average expected error, $\epsilon\overline{E}_{max}/\overline{n}$, and the node is followed by a non-adjacent tuple pair or at least δ adjacent tuples. Once the whole ITA relation has been processed we know the real maximal error that can be made, E_{max}. Therefore, as long as the total error introduced so far, E_{tot}, does not exceed the error bound we use GMS to finalize the merging process. The worst-case time and space complexity of gPTA_ϵ is the same as of gPTA_c.

Example 4.6. We run gPTA_ϵ on the **proj** relation with $\epsilon = 0.5$ and $\delta = 1$. We set $\overline{E}_{max} = E_{max}$ of the corresponding ITA result, which is 269 285.714, and $\overline{n} = n = 7$. Therefore, if we were to merge all the tuples, the average error we would make per step would be $\epsilon E_{max}/n = 19\,234.69$. gPTA_ϵ reads the ITA tuples one by one and tries to merge those that introduce less than the expected average error. The first such candidates are s_2, s_3 (see Figure 4.1(b)). However, the δ parameter dictates to read the tuple s_4 before merging that pair.

THEOREM 4.3. *The output of the* gPTA_ϵ *algorithm with* $\delta = \infty$ *is identical to that of GMS if* $\frac{\overline{E}_{max}}{\overline{n}} \leq \frac{E_{max}}{n}$.

Proof. The gPTA_ϵ algorithm will consider merging a pair of tuples s_i, s_{i+1} only if $dsim(s_i, s_{i+1}) \leq \epsilon\frac{\overline{E}_{max}}{\overline{n}} \leq \epsilon\frac{E_{max}}{n}$. Therefore, as follows from Proposition 4.3, the algorithm will merge the same tuple pairs that GMS would. □

The estimated values $\overline{n}$ and $\overline{E}_{max}$ play an important role in the gPTA_ϵ algorithm. First, the estimated values may influence the correctness of the final output. Second, the precision of the estimate affects the size of the heap. The estimation of the ITA result size is easy, since it can be at most twice as large as the argument relation, thus $\overline{n} = 2|\mathbf{r}| - 1$. Estimating the maximal error, $\overline{E}_{max}$, is more complicated. The key aspect to consider here is how precise should the estimate be. As long as $\overline{E}_{max} \leq E_{max}$, the estimate only influences the size of the heap. That is, when $\overline{E}_{max} \ll E_{max}$, none or very few early merges will take place. Thus, the heap will be filled with almost the entire ITA result before the merging will commence. On the other hand, when the error is overestimated, i.e., $E_{max} < \overline{E}_{max}$, we cannot guarantee that the result is the same as for GMS. It seems reasonable to sample the argument relation and compute the corresponding ITA result to

```
1  Algorithm: gPTA_ε(r, A, F, δ, ε)
2  H ← new empty heap;
3  Initialize the ITA operator with F, A, and r;
4  LastGapId ← 0; BG ← 0; AG ← 0;
5  E_tot ← 0; E_max ← 0;
6  Estimate Ē_max and n̄;
7  while s_i ← next tuple from G^ITA[A, F]r do
8      Lines 6 – 12 from gPTA_c algorithm in Algorithm 4.1;
9      while true do
10         N ← PEEK();
11         if N.key > εĒ_max/n̄ then break;
12         if N.id < LastGapId then
13             BG ← BG − 1;
14             E_tot ← E_tot + N.key;
15             MERGE();
16         else if N.id > LastGapId ∧ N has δ successors then
17             AG ← AG − 1;
18             E_tot ← E_tot + N.key;
19             MERGE();
20         else
21             break;
22 while true do
23     N ← PEEK();
24     if N.key < ∞ ∧ (E_tot + N.key)/E_max ≤ ε then
25         MERGE();
26         E_tot ← E_tot + N.key;
27     else
28         break;
29 return H;
```

Algorithm 4.2: Greedy algorithm, gPTA_ϵ, for the error-bounded PTA.

Table 4.1: The aggregation queries used for evaluation of PTA

(a) ETDS

Name	Grouping	Agg. Functions	ITA Size	c_{min}
E1	–	avg	6 394	1
E2	–	max	6 394	1
E3	–	sum	6 394	1
E4	Emp.No., Dep.	avg	5 419 493	339 067

(b) Incumbents

Name	Grouping	Agg. Functions	ITA Size	c_{min}
I1	Dep., Proj.	avg	16 144	131
I2	Dep., Proj.	max	16 144	131
I3	Dep., Proj.	sum	16 144	131

(c) Time series

Name		Dimensionality	ITA Size	c_{min}
T1	Chaotic	1	1 800	1
T2	Tide	1	8 746	1
T3	Wind	12	6 574	216

(d) Synthetic

Name	Grouping	Dimensionality	ITA Size	c_{min}
S1	–	10	10 000 000	1
S2	yes	10	10 000 000	50 000

obtain an error estimate. A more detailed investigation of this aspect is part of the future work.

4.6 Experimental Evaluation

In this section we evaluate experimentally the reduction quality and scalability properties of the greedy algorithms. Using the same datasets and aggregation queries as in Chapter 3 we quantify the error that they induce and compare it to other related approaches. The summary of these queries is reprinted in Table 4.1. Full description of each dataset and query is available in Section 3.6.1.

We implemented the algorithms introduced in this paper as well as ATC [8], APCA [15], DWT [40], and PAA [31] using Java™ version 6. They are executed on a Linux machine with 4 AMD 2600MHz Opteron processors and 16GB of RAM. An Oracle 11g database running on the same machine is used as the data storage medium.

4.6.1 Reduction Quality

We quantify the reduction error of the greedy PTA algorithms. First we find out how close greedy approximation and the optimal solution are as well as whether the former can outperform other known data approximation algorithms. For this purpose we use $\mathrm{gPTA_c}$ algorithm with $\delta = \infty$ as the representative of the greedy merging strategy. Note that gPTA_ϵ would yield an identical result. Next we evaluate the influence of δ parameter on the results of both, $\mathrm{gPTA_c}$ and gPTA_ϵ, algorithms.

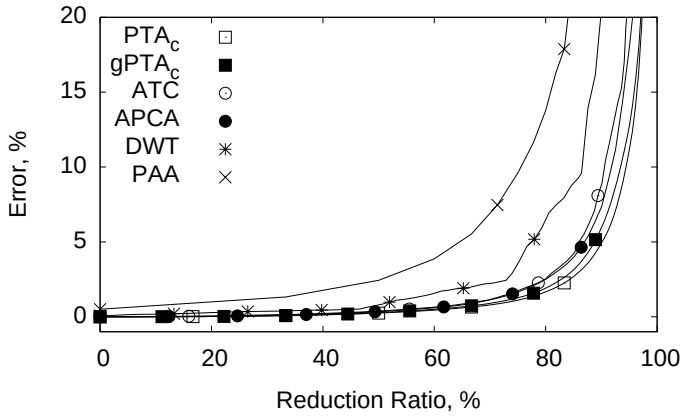


Figure 4.5: Reduction errors induced by different algorithms

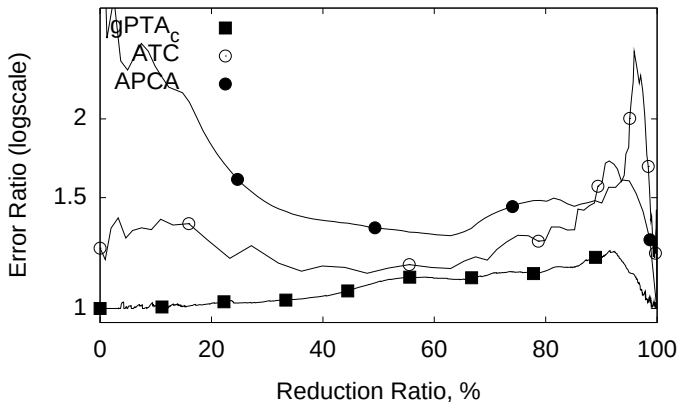


Figure 4.6: Reduction error ratio of different algorithms

As before we measure the error induced by the algorithms at each size bound and Figure 4.5 depicts the resulting error growth curves for T1 dataset. Observe that the curve induced by $\mathrm{gPTA_c}$ is the closest to the optimal result obtained from $\mathrm{PTA_c}$, indicating that the greedy algorithm is the

closest to the optimal result. On the other hand, DWT and PAA approaches introduce significantly more error. Figure 4.6 offers an alternative view of this experiment. Here we take $\mathrm{PTA_c}$ result as a baseline and plot the ratio of reduction error to the baseline. Thus an error ratio value 1 means that the approach produced optimal reduction and any greater value signifies divergence from it. The ratio of greedy reduction error remains very close to 1 throughout and increases slightly with the number of merging steps to reach a maximum of 1.25. This behavior is predicted by Theorem 4.1. ATC and APCA lag behind. DWT and PAA algorithms perform significantly worse and are not included in the graph.

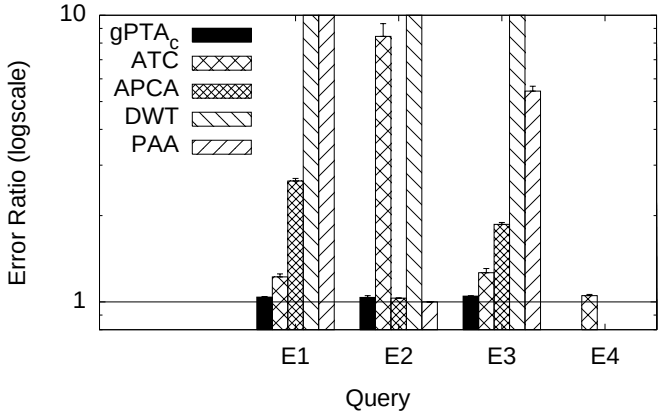


Figure 4.7: Average reduction error ratios induced by various algorithms over the ETDS dataset

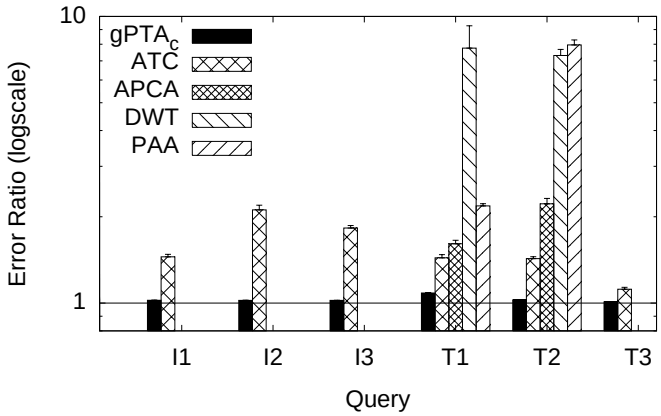
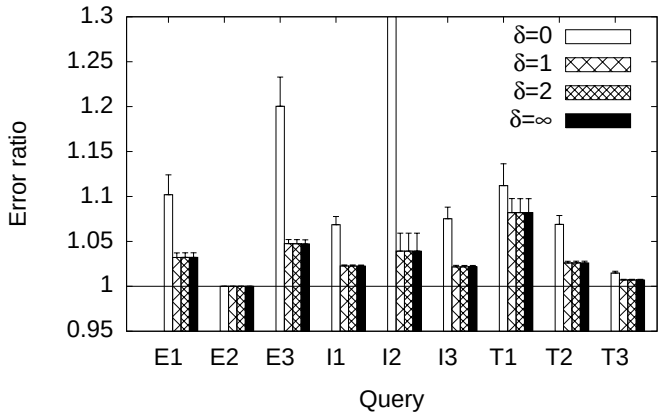
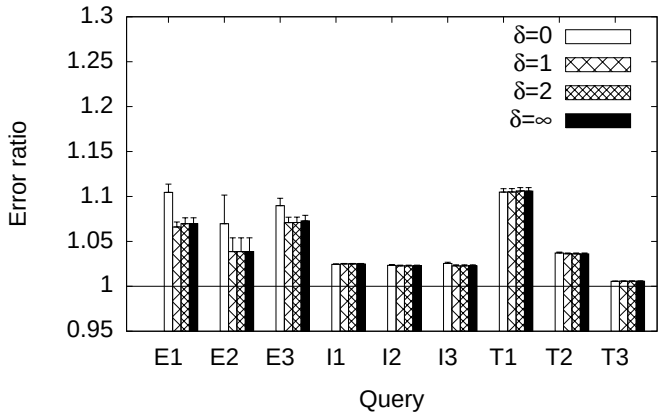


Figure 4.8: Average reduction error ratios induced by various algorithms over the Incumbents and time series datasets

We run the same experiment on all the datasets and summarize the



(a) $\mathrm{gPTA_c}$



(b) gPTA_ϵ

Figure 4.9: The impact of δ parameter

result in Figures 4.7 and 4.8. For each query the average reduction error ratio over the full range of c values is shown. A pattern-filled bar depicts the average ratio and a thin line represents the standard error. APCA, DWT, and PAA algorithms cannot be used for queries I1, I2, I3, and T3 as they have not been defined for data with multiple aggregation groups or temporal gaps. For query E4 we take GMS as the baseline and compare the error of ATC to it because E4 is too large to be evaluated with $\mathrm{PTA_c}$. The value of the ratio is above one, indicating that GMS outperforms ATC. Overall the $\mathrm{gPTA_c}$ algorithm consistently provides the best error ratio, i.e. its results are closest to those of $\mathrm{PTA_c}$. ATC is the second best algorithm, however, its performance is not consistent. ATC shows satisfactory results for queries E4 and T3, but not for I1 and I2.

Now we evaluate the influence of the δ parameter to the correctness

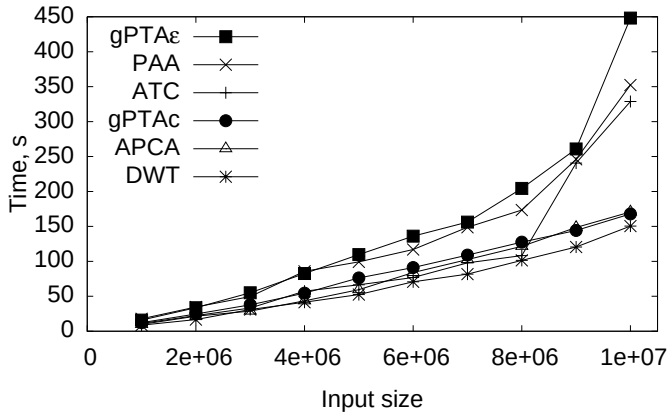


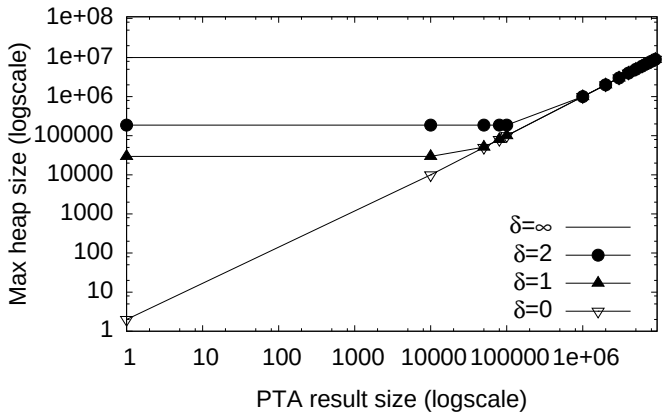
Figure 4.10: Performance of greedy algorithms compared to other linear approximation methods

of the final result of $\mathrm{gPTA_c}$ and $\mathrm{gPTA_\epsilon}$. Figure 4.9 shows the average error ratio (and standard error) of both algorithms with varying δ and different datasets. As before, the average is computed over errors induced by varying the size bound c and error bound ϵ respectively. When $\delta = 0$ the algorithms are at their worst. When $\delta = \infty$ the algorithms return the best possible result, i.e. equivalent to that of the greedy merging strategy. However, we see that for $\delta \geq 1$ the results are practically the same. In the following experiments we will show that setting δ reduces the heap significantly yielding better running time performance.

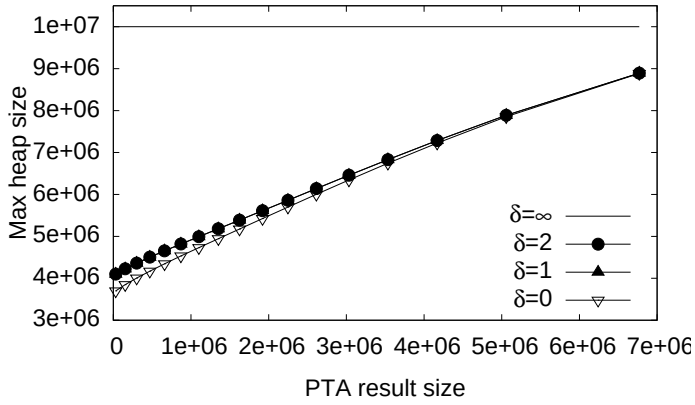
4.6.2 Scalability Evaluation

We compare the performance of $\mathrm{gPTA_\epsilon}$ and the $\mathrm{gPTA_c}$ ($\delta = 1$) algorithms to APCA, DWT, PAA and ATC. Observe that $\mathrm{gPTA_\epsilon}$ and $\mathrm{gPTA_c}$ are different and should yield different performance figures. We use subsets of the synthetic dataset without temporal gaps ranging in size from 1 to 10 million tuples. The size bound is set to 10% of the input. Satisfying such a size bound means a significant approximation effort for all algorithms. Figure 4.10 depicts the average running time of each algorithm with respect to the size of the input. The $\mathrm{gPTA_\epsilon}$ is the slowest and rapidly diverges from the others as it has to deal with an ever-increasing heap structure. On the other hand, $\mathrm{gPTA_c}$ is very fast and loses out only to DWT. Such a performance advantage is due to the significantly smaller priority queue that $\mathrm{gPTA_c}$ operates on. With the following experiment we verify this claim.

We measure the heap size by running $\mathrm{gPTA_c}$ on the whole synthetic relation and varying the size of the output, c, as well as the merging bound, δ. As already mentioned, the $\mathrm{gPTA_c}$ algorithm operates on a heap of size



(a) $\mathrm{gPTA_c}$



(b) gPTA_ϵ

Figure 4.11: Maximal heap size of $\mathrm{gPTA_c}$ and gPTA_ϵ algorithms as a function of output size

$c + \beta$ where β varies throughout the aggregation, depending on the distribution of the data and the value of c. For each query we measure the maximum value of β. Intuitively, β is maximal when $c = 1$ and the tuples in the dataset are very similar. In such situations the most similar pair can often be the most recently read one and thus more ITA tuples have to be read.

Figure 4.11 illustrates the result. The horizontal axis ranges over the values of c and the vertical over β. As expected, the maximum values are experienced when $c = 1$ and then decrease rapidly with the change of c. The bigger the value of δ, the more space is needed for the queue. When $\delta = 0$ the queue needs only $c + 1$ space. In the limit, when $\delta = \infty$ and data has no temporal gaps, the space needed is equal to the size of the input.

Most importantly, this experiment shows that $\mathrm{gPTA_c}$ uses only a fraction of space in comparison to other methods that have to store the whole input (10 000 000 tuples) in memory.

In summary, the $\mathrm{gPTA_c}$ and $\mathrm{gPTA_\epsilon}$ algorithms show very good reduction quality outperforming other linear data approximation methods consistently and significantly. In addition, they scale remarkably well for huge datasets.

4.7 Summary

When efficient performance in the evaluation of PTA queries is the top priority, the proposed greedy algorithms, $\mathrm{gPTA_c}$ and $\mathrm{gPTA_\epsilon}$, should be used. We proved that the error ratio of greedy approach to the optimal one is upper-bounded by $O(\log n)$, where n is the size of corresponding ITA relation. The algorithms use $O(c + \beta)$ space and $O(n \log(c + \beta))$ time to obtain an output of size c, and β is typically very small. Experimental evaluation confirmed theoretical estimations and demonstrated that the greedy algorithms scale very well for large datasets providing a significantly better approximation quality than other comparable approximation approaches.

CHAPTER 5

Temporal Ranking

Since the beginning of the world wide web, interlinked sources of information have been the subject of active research in the knowledge discovery community. Several approaches have been proposed to deal with the challenging problems of information summarization and ranking using link analysis. In this chapter, we leverage from these previous approaches and focus on the problem of ranking aggregated news stories within their historical context by exploiting their content similarity. We observe that news stories evolve and we rank them in a time and query dependent manner. We do this in two steps. First, the *mining* step discovers metastories, which constitute meaningful groups of similar stories that occur at arbitrary points in time. Second, the *ranking* step uses well known measures of content similarity to construct implicit links among all metastories, and uses them to rank those metastories that overlap the time interval provided in a user query. We use real data from conventional and social media sources (weblogs) to study the impact of different meta-aggregation techniques and similarity measures in the final ranking. We evaluate the framework using both objective and subjective criteria, and discuss the selection of clustering method and similarity measure that lead to the best ranking results.

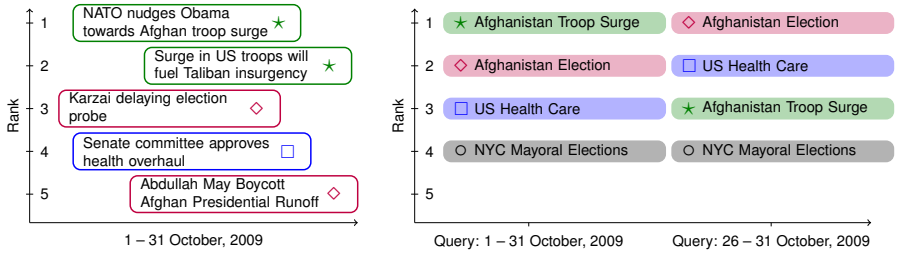
5.1 Introduction

Existing technologies for online news browsing allow users to have continuous access to up-to-date news, as well as to a wide range of related opinions and comments coming from the social media. However, the larger-scale problem of aggregating, searching, and ranking such content within its historical context has not been thoroughly addressed. Currently, news search engines evaluate the relevance of a story based only on its content and how well it matches the keywords specified in the query. However, the temporal evolution of the story, and its behavior within the ecosystem of all stories changing within the time interval of interest is an important relevance factor that has thus far not been exploited. In addition, news aggregation engines break the evolving stories into smaller, highly consistent, short-lived clusters of similar news articles. Multiple events within a single story typically generate distinct clusters as the content of the newer articles drifts away from that of existing older clusters. Consequently, events that are widely spaced in time will be assigned to separate clusters even if they belong to the same storyline. Under these conditions, a keyword based search should retrieve the separate clusters, and, very importantly, will rank them separately in terms of their content and how well they match the query.

In this chapter we show that the temporal relationships between evolving stories should affect their rank, and how a temporally sensitive ranking can be computed. We propose that the first step towards a meaningful temporally-sensitive ranking of stories is the aggregation of separate events related to the same story into a single cluster, here called a *metastory*. The generation of metastories is a challenging problem because even though the central theme of each metastory is the same, the actual content of the news articles within can vary widely. The second step, once the metastories have been generated, is the computation of a temporally sensitive rank by considering only those metastories whose evolution overlaps the time interval specified by the user in the query. The two steps together yield a novel framework that facilitates ranking of the metastories by relevance even in the absence of query keywords. All that is needed is the time interval of interest.

More specifically, our framework consists of two independent components. First, the *metastory mining* step employs a clustering method to create metastories by aggregating a set of story clusters, similar to the way news aggregators place stories together. Our stories are provided by a state-of-the-art platform for news and social media aggregation[1], which contains not only news articles, but also blogs related to the stories. The inclusion of social media is critical in allowing us to evaluate the quality of

[1] `http://www.thoora.com`



(a) Individual stories (b) Metastories

Figure 5.1: Effect of the query time interval on the ranking results

the ranking produced by our method. The mining process is purely content driven, and allows for the generation of metastories spanning large time intervals. We evaluate several existing clustering techniques [56, 21, 49], as well as different similarity measures in order to determine the best possible combination for this step.

In the second component, the *ranking* step, we proceed to rank a given set of metastories with regard to the time interval specified by the user. Only metastories active within this interval are considered and, very importantly, only the metastory articles that were active within the interval are used to determine relevance. Ranking is done through link analysis. However, unlike traditional PageRank [13], which is based on hypertext links, we construct implicit links using content-based similarity between all pairs of active metastories. We exploit the symmetry of similarity measures to avoid the expensive power iteration traditionally used to compute the PageRank vector. We study the effect of using different similarity measures to construct the implicit links between metastories, and select the measure that yields the best ranking based on both objective and subjective evaluation results.

The temporal ranking framework in this chapter is presented and evaluated in the context of news aggregation. However, it is straightforward to apply it in the context of general temporal aggregation. The framework would first discover the aggregation groups and, second, allow to disseminate the most important ones. Next, we provide a short example that we use throughout the chapter. Further on, we formally define the framework and, finally and most importantly, we provide extensive evaluation of the framework using real-world data.

5.2 Example

In Figure 5.1 we show an example of the news story aggregation and ranking process. Individual story elements, shown in Figure 5.1(a), were gathered during the month of October 2009. The figure shows only 5 out of the 700 news-related clusters that were used. Even so, notice that the individ-

ual clusters can be grouped into metastories corresponding to important events occurring during the month. Metastory aggregation is performed once, over the set of all existing story components, and independently of any interval a user may be interested in.

Once metastories have been constructed, the ranking step determines their relative ordering with regard to a time interval specified by the user. Figure 5.1(b) shows two sample rankings for different query intervals: Oct. 1-31, and Oct. 26-31, 2009[2]. Notice that the relative ranking of metastories changes. In particular, the "Afghanistan Troop Surge" metastory is the most important overall story of October 2009, but when only the last week of the month is considered its relevance is lower compared to other metastories that were more active during this specific period. Such relationships can only be discovered through meta-aggregation and time-dependent ranking. Given the same set of metastories, a content-based search would produce the same ranking independent of the period in which the user was interested. In summary, the time-dependent nature of the ranking process, coupled with the more comprehensive scope of metastories, yields a more informative and representative ranking of news events.

5.3 The Framework

In this section we formally describe our ranking framework. First, we define our basic building block – a story. Next, we describe how metastories are constructed from similar stories and propose how content-based similarity measures can be used to rank metastories. Finally, we discuss how the ranking of metastories is made time-dependent by considering a specified interval, which can be provided as part of a user's query.

In our particular application scenario, a story refers to a cluster of news and blog articles. Thus, a story is represented by the term frequency vector and the *lifespan* associated with it is defined as the interval bounded by two time points, t_b and t_e, denoting the timestamps of the earliest and latest documents in the story. If for two lifespans, T' and T'', we have $T' \cap T'' \neq \emptyset$, then we say that they overlap (or intersect).

DEFINITION 5.1. *Let* $\mathbf{W}$ *be a keyword dictionary. A* story s *is defined as a pair* $s = \langle f_s, T_s \rangle$*, where* f_s *is the term frequency vector whose entries corresponding to particular terms* $w_i \in \mathbf{W}, i = 1, \ldots, |\mathbf{W}|$ *represent the number of times the term* w_i *occurs in all documents within* s*.* $T_s = [t_{b_s}, t_{e_s}]$ *is the lifespan associated with* s*.*

Example 5.1. Table 5.1 shows a few of the stories used in the example in Figure 5.1(a) together with a snapshot of their keywords and corresponding

[2]Without loss of valuable information, we use the most descriptive titles for the stories and metastories in Figure 5.1.

lifespans. Stories are listed in the same order as in the figure, e.g., story s_1 corresponds to the title "NATO nudges Obama towards Afghan troop surge decision". Within that story the keyword "nato" appears 808 times. Since the stories themselves are clusters of a large number of related documents, keyword frequencies can be quite high.

Table 5.1: Snapshot of story keywords

s	Keywords, f_s			Lifespan, T_s
s_1	nato: 808	afghan: 717	troop: 642	22 – 27/10
s_2	afghan: 3472	obama: 3200	troop: 3027	8 – 15/11
s_3	afghan: 9917	elect: 9891	karzai: 7093	11 – 25/10
s_4	health: 12856	care: 11147	obama: 9049	10 – 30/10
s_5	afghan: 6505	karzai: 6398	hamid: 5633	30/10 – 5/11

5.3.1 Constructing Metastories

Intuitively, a *metastory* is a collection of stories with associated notions of lifespan and term frequency vector. Given as input the story set, $\mathbf{S}$, a clustering algorithm partitions it into k disjoint non-empty subsets, $\mathbf{S}_1, \ldots, \mathbf{S}_k \subseteq \mathbf{S}$, such that $\bigcup_{i=1,\ldots,k} \mathbf{S}_i = \mathbf{S}$ and $\bigcap_{i=1,\ldots,k} \mathbf{S}_i = \emptyset$. Each subset represents a distinct metastory. In Section 5.4, we provide details about the clustering algorithms used in our study.

DEFINITION 5.2. *Given a set of stories* $\mathbf{S}$*, a* metastory m *is a triplet* $m = \langle \mathbf{S}_m, f_m, T_m \rangle$*, where*

$$
\begin{aligned}
&(a) \qquad \mathbf{S}_m \subseteq \mathbf{S}, \\
&(b) \qquad f_m = \oplus_{s \in \mathbf{S}_m}(f_s), \\
&(c) \qquad T_m = [\min_{s \in \mathbf{S}_m}\{t_{b_s} | t_{b_s} \in T_s\}, \max_{s \in \mathbf{S}_m}\{t_{e_s} | t_{e_s} \in T_s\}],
\end{aligned}
$$

and $\oplus$ *is an aggregation operator over a set of frequency vectors.*

In the above definition the lifespan T_m of the metastory S_m refers to the maximum time-interval covering the entire set of lifespans associated with all stories in S_m. The term frequency vector f_m for S_m is obtained through an operator $\oplus_{s \in S_m}(f_s)$ that performs aggregation over a set of term frequency vectors. Some possible choices for aggregation operators are:

$$\mathbf{Sum}: \qquad \sum_{s \in \mathbf{S}_m} f_s, \tag{5.1}$$

$$\mathbf{Mul}: \qquad \prod_{s \in \mathbf{S}_m} f_s, \tag{5.2}$$

$$\mathbf{Max}: \qquad f_{\arg\max\{size(s) | s \in \mathbf{S},\}} \tag{5.3}$$

where $size(s)$ is the number of documents (blogs and news) found in the story s. The Sum and Mul operators summarize all term frequency vectors into one. The Max operator, on the other hand, chooses the term frequency vector of the biggest story in S_m as the representative of S_m.

Example 5.2. From the stories depicted in the previous example we construct metastories as shown in Table 5.2. Metastory m_1 is formed by the stories s_1 and s_2. It corresponds to title "Afghanistan Troop Surge" in Figure 5.1(b). Similarly, the titles of m_2 and m_3 are, respectively, "Afghanistan Election" and "US Health Care". Observe that in reality the size of a metastory can vary widely. When a metastory covers only a short-lived event, it is likely to contain just one or few stories. Some metastories, such as those related to the war in Afghanistan, cover long time intervals and consequently contain thousands of stories.

Table 5.2: Clustering stories yields metastories

	S_m	Keywords, f_m				Lifespan, T_m
m_1	$\{s_1, s_2\}$	afghan: 4189	troop: 3669	obama:3200	nato: 808	22/10 – 15/11
m_2	$\{s_3, s_5\}$	afghan:16422	karzai:13491	elect:9891	hamid:5633	11/10 – 5/11
m_3	$\{s_4\}$	health:12856	care:11147	obama:9049		10/10 – 30/10

5.3.2 Ranking Metastories

In this section we demonstrate how to rank metastories using their content similarity. To do it we assume that the set of stories, S, and the set of metastories, M, are given. We also assume a content-based similarity measure over M. The measure is not necessarily the same as the one used by the clustering method. Without loss of generality, we assume frequency vectors represent proper probability distributions, i.e. they are normalized to have unit mass.

DEFINITION 5.3. *Let $\mathbf{M}$ be the set of metastories and let $sim : M \times M \to [\epsilon, 1]$ be a content-based similarity function with $\epsilon > 0$[3]. The* rank *of metastory $m \in \mathbf{M}$ is defined as:*

$$rank(\mathbf{M}, m) = \sum_{m' \in \mathbf{M}} \frac{sim(m, m')}{\sum_{m'' \in \mathbf{M}} sim(m', m'')} \cdot rank(M, m'). \tag{5.4}$$

The definition states that a metastory should be ranked high if it is similar to other highly ranked metastories. The ranking step of our framework can be seen as a random walk similar to that of PageRank [13], where

[3]We assume that $\epsilon > 0$ is the minimum possible similarity value between metastories in order to ensure that the graph constructed through the similarity values is fully connected

the transition probabilities depend on metastory similarities. Therefore, the rank vector for all metastories in $\mathbf{M}$ corresponds to the stationary distribution of a random walk over the graph whose nodes correspond to metastories, and whose edges are the normalized similarities. While the standard way of computing the stationary distribution requires the use of the power iteration method, we show through the following lemma that if the similarity measure is symmetric, the computation is straightforward and efficient. The lemma follows [16] in showing that the value of $rank(\mathbf{M}, m)$ is proportional to the sum of the similarities from m to all metastories in $\mathbf{M}$.

LEMMA 5.1. *Let $\mathbf{M}$ and sim be as in Definition 5.3. If sim is symmetric, i.e. for every $m', m'' \in \mathbf{M}$, we have $sim(m', m'') = sim(m'', m')$, then for some constant λ the following holds:*

$$rank(\mathbf{M}, m) = \lambda \sum_{m' \in \mathbf{M}} sim(m, m'). \tag{5.5}$$

Proof. We rewrite Equation 5.4 using matrix notation. Let $\mathbf{A}_{k \times k}$ be the similarity matrix such that $\mathbf{A}_{ij} = sim(m_i, m_j)$. By definition, $\mathbf{A}$ is symmetric and represents a fully connected, undirected graph. Let the vector $\mathbf{d}$ be the row-wise sum of values in $\mathbf{A}$,

$$\mathbf{d}_i = \sum_{j=1,\ldots,k} \mathbf{A}_{ij}. \tag{5.6}$$

Let $\mathbf{D} = diag(\mathbf{d})$ be a diagonal matrix with $\mathbf{d}$ in its main diagonal. One can easily verify that Equation 5.4 is equivalent to

$$rank(\mathbf{M}, m) = \mathbf{A} \cdot \mathbf{D}^{-1} rank(\mathbf{M}, m). \tag{5.7}$$

Since $\mathbf{A}$ is strictly positive and symmetric, $\mathbf{A} \cdot \mathbf{D}^{-1}$ is a stochastic and irreducible matrix and $rank(\mathbf{M}, m)$ is its right eigenvector. For such a matrix it is known that its greatest eigenvalue is one and the space of eigenvectors for this eigenvalue has dimension 1. In [16] it is shown that $\mathbf{d}$ is also the eigenvector of $\mathbf{A} \cdot \mathbf{D}^{-1}$ for the eigenvalue 1. Since $\mathbf{D}^{-1} \cdot \mathbf{d} = \mathbf{1}$ and $\mathbf{A} \cdot \mathbf{1} = \mathbf{d}$, we have

$$\mathbf{d} = (\mathbf{A} \cdot \mathbf{D}^{-1}) \cdot \mathbf{d} \tag{5.8}$$

$$= \mathbf{A} \cdot \mathbf{1} \tag{5.9}$$

$$= \mathbf{d}. \tag{5.10}$$

Therefore $rank(\mathbf{M}, m_i)$ is in the space generated by $\mathbf{d}$ and hence is a multiple of $\mathbf{d}$, that is, $rank(\mathbf{M}, m_i) = \lambda \mathbf{d}_i$. Since $rank(\mathbf{M}, m_i)$ should be a vector with nonnegative values and norm 1, we have $\lambda = 1/\|\mathbf{d}\|$. □

The preceding lemma proves that it is not necessary to perform the costly power iteration procedure for symmetric similarity matrices. Instead, an identical ordering of metastories can be obtained by computing the column or row-wise sum of similarity values in the adjacency matrix $\mathbf{A}$.

Example 5.3. Continuing the previous example we compute pairwise χ^2 distances between the normalized keyword distributions of metastories in Table 5.2. Converting them to similarities we obtain the adjacency matrix in Table 5.3 where the rank value of a metastory corresponds to a row-wise sum. It is important to note that ϵ here is so small that it cannot influence the final ranking.

Table 5.3: Metastory similarity matrix and the resulting ranks

	m_1	m_2	m_3	$rank$
m_1	1	0.36	0.33	1.69
m_2	0.36	1	ϵ	$1.36+\epsilon$
m_3	0.33	ϵ	1	$1.33+\epsilon$

5.3.3 Time Sensitive Ranking of Metastories

Having considered the general ranking problem, we now incorporate the temporal dimension defined by the user query into the ranking of metastories. The main challenge here is that the ranking of the metastories that intersect with the user-specified query should depend only on those member stories that also intersect the query. This way the temporal aspect associated with a specific query captures the evolution of stories as illustrated previously in Figure 5.1.

In order to define the query dependent ranking, we first introduce the *reduction operator*. Given a metastory the operator constructs a new metastory that contains only the stories intersecting the query interval. It calculates the corresponding keyword frequency vector and a valid lifespan. Naturally, some reduced metastories may contain no stories.

DEFINITION 5.4. *Let* $\mathbf{S}$ *be the set of stories,* $\mathbf{M}$ *be the set of metastories, and* $Q = [t_{bQ}, t_{eQ}]$ *be the user query. For a metastory* $m \in \mathbf{M}$*,* $m = \langle \mathbf{S}_m, f_m, T_m \rangle$ *the operator* $reduce(Q, m)$ *constructs a new metastory* $m' = \langle \mathbf{S}_{m'}, f_{m'}, T_{m'} \rangle$ *such that*

$$
\begin{aligned}
&(a) \qquad \mathbf{S}_{m'} \subseteq \mathbf{S}_m \wedge \forall s \in \mathbf{S}_{m'}(T_s \cap Q \neq \emptyset),\\
&(b) \qquad f_{m'} = \oplus_{s \in \mathbf{S}_{m'}}(f_s),\\
&(c) \qquad T_{m'} = [\min_{s \in \mathbf{S}_{m'}}\{t_{bs} | t_{bs} \in T_s\}, \max_{s \in \mathbf{S}_{m'}}\{t_{es} | t_{es} \in T_s\}].
\end{aligned}
$$

Only the metastories that are relevant to the user query should get a rank, therefore, the query dependent rank of a metastory is zero if its reduced version contains no stories. Otherwise, it is equal to the rank of the reduced version in the set of all reduced and non-empty metastories.

Algorithm: $\mathrm{TRANK}(\mathbf{S}, \mathbf{M}, Q)$
$rank_Q(\mathbf{M}, m) \leftarrow 0, \forall m \in \mathbf{M}$;
$\mathbf{M}_Q \leftarrow \{m' | m' = reduce(Q, m) \wedge m \in \mathbf{M} \wedge \mathbf{S}_{m'} \neq \emptyset\}$;
for $m \in \mathbf{M}$ **do**
  $m' \leftarrow reduce(Q, m)$;
  if $m' \in \mathbf{M}_Q$ **then**
    $rank_Q(\mathbf{M}, m) \leftarrow \sum_{m'' \in \mathbf{M}_Q} sim(m', m'')$;
return $rank_Q$;

Algorithm 5.1: The temporal ranking algorithm, TRANK

DEFINITION 5.5. *Let $\mathbf{S}, \mathbf{M}$, and Q be as in Definition 5.4. Let $\mathbf{M}_Q$ be a set such that $\mathbf{M}_Q = \{m' | m' = reduce(Q, m) \wedge m \in \mathbf{M} \wedge \mathbf{S}_{m'} \neq \emptyset\}$. Then the* query dependent rank *of a story $m \in \mathbf{M}$ is $rank_Q(\mathbf{M}, m) =$*

$$= \begin{cases} rank(\mathbf{M}_Q, reduce(Q, m)), & \text{if } reduce(Q, m) \in \mathbf{M}_Q, \\ 0, & \text{otherwise.} \end{cases}$$

Example 5.4. Consider a temporal ranking query for the period between the 26^{th} and 31^{st} of October, 2009. The reduced set of example metastories is shown in Table 5.4. The metastories m_1, m_2 are reduced to only one story each, while metastory m_3 remains intact.

Table 5.4: Metastories between 26^{th} and 31^{st} of October, 2009

	S_m	Keywords, f_m			Lifespan, T_m
m_2	$\{s_5\}$	afghan: 6505	karzai: 6398	hamid:5633	30/10 – 5/11
m_3	$\{s_4\}$	health:12856	care:11147	obama:9049	10 – 30/10
m_1	$\{s_1\}$	nato: 808	afghan: 717	troop: 642	22 – 27/10

The algorithmic procedure needed to evaluate the ranks of metastories in the set $\mathbf{M}$ given a query Q is shown in Algorithm 5.1 and directly follows the above definition. We start by assigning rank zero to each metastory. Then the set of reduced and non-empty metastories, $\mathbf{M}_Q$, is computed. The pairwise similarities of these metastories are used to compute their query dependent ranks. Finally, we note that it is possible to implement our framework in a more efficient way by using specialized data structures such as the SB-Tree [60].

5.4 Experimental Evaluation

In order to design an optimal temporal ranking framework we must make two crucial choices. First, we have to select a clustering algorithm that produces good metastories from the underlying stories. Second, we have to

choose a similarity measure that ranks the metastories best. As mentioned earlier, the similarity measure used by the clustering algorithm to discover the metastories does not necessarily have to be the same as the one used for ranking.

In this section, we conduct a thorough experimental study in order to provide sufficient information regarding these two choices and to illustrate the usefulness of our approach. We will evaluate different clustering algorithms in terms of the quality of content in the resulting metastories, and we discuss the potential tradeoffs between speed and quality in the clustering process. Once we have determined an appropriate clustering method, we study the difference that the various similarity measures make in the ranking result.

5.4.1 Datasets

For the evaluation we use the data acquired through the online news aggregation platform. We have collected story clusters over a period of 56 days starting from 15 September 2009. The story clusters are classified into seven categories: Business, Entertainment, Controversy, Lifestyle, Politics, Science & Technology, and Sports. Within any given category, blog posts and news articles can only be assigned to a single story cluster. For each category we create a separate dataset that records the 50 top-ranked stories for each day. The rank of a story cluster is based on how much buzz the corresponding story generates, which is measured by the amount of activity in the blogosphere and on Twitter regarding the story. Typically, the list of top stories in consecutive days does not change significantly as new hot stories enter the list and stale stories become irrelevant. The resulting datasets contain an average of 700 unique stories per category, spanning the 56 day period.

5.4.2 Evaluation of Clustering Algorithms

We compare three different clustering methods: efficient graph-based segmentation (ES) [21], Markov Clustering (MCL) [56] and the agglomerative Information Bottleneck (aIB) [49]. These methods represent a wide range of clustering paradigms. ES uses a simple cluster growing approach based on minimum spanning trees, MCL is a good representative of stochastic clustering methods, and aIB is based on information theory. They have also been applied successfully in different fields. ES has been used extensively for image analysis [21], MCL has been successfully applied in protein analysis [19, 24], and aIB has proven itself as a successful method for clustering database and document collections [49], while its extension, LIMBO, can be directly applied to the problem of online clustering of streaming data [5].

The ES method, the fastest of the three, is based on the intuition that a good clustering is such that the cohesion between elements within a cluster must be high, while the coupling between elements in different clusters must be low. Data is represented by a weighted graph where the edge weights are given by the dissimilarity between pairs of data entities. By comparing and merging neighboring regions in the graph the algorithm produces a final set of clusters efficiently.

The MCL algorithm is based on the simulation of stochastic flow over a graph representing the data. Due to the properties of Markov processes, strongly connected components corresponding to clusters will be characterized by edges with strong probability flow, whereas the flow between nodes that correspond to data from different clusters will be weak. MCL uses a probability diffusion process followed by edge filtering to remove weak edges and enhance the flow within clusters.

For the ES and MCL algorithms we use the χ^2 distance, as it has been shown to provide close to optimal performance at a low computational cost in problems that involve discriminating between probability distributions [47]. Given two stories s_i and s_j with their respective keyword frequency distributions f_{s_i}, and f_{s_j}, the χ^2 distance between the stories is defined as [47]

$$\chi^2(s_i, s_j) = \sum_{w \in \mathbf{W}} \frac{(f_{s_i}(w) - \hat{f}(w))^2}{\hat{f}(w)}, \tag{5.11}$$

where $\hat{f}(w) = \frac{1}{2} f_{s_i}(w) + \frac{1}{2} f_{s_j}(w)$ is the average keyword frequency distribution. In this case, we consider $f_{s_i}(w)$ and $f_{s_j}(w)$ to be properly normalized and hence weight them by $\frac{1}{2}$.

For the aIB algorithm, the input is modeled as a joint probability distribution of two variables. One variable captures the objects to be clustered, while the other represents the values over which these objects are defined. In our case, these are the story clusters and the set of keywords in the stories, respectively. aIB defines the notion of *information loss* (IL) using the aforementioned joint probability distribution to quantify distance between clusters. It iteratively merges pairs of clusters (initially individual data points) with the minimum information loss to form metastories. This process continues until a certain quality in the clustering is achieved or a desired number of clusters is discovered.

The information loss (IL) is defined through the *Jensen-Shannon* (JS) divergence in [49]. For two stories s_i and s_j, represented by their respective normalized keyword frequency distributions f_{s_i}, f_{s_j}, the *Jensen-Shannon (JS)* divergence is defined as

$$JS_{\mathbf{\Pi}}(s_i, s_j) = H(\pi_1 f_{s_i} + \pi_2 f_{s_j}) - \pi_1 H(f_{s_i}) - \pi_2 H(f_{s_j}), \tag{5.12}$$

where $H(f) = \sum_w f(w) \log f(w)$ is the entropy of the words $w \in W$ in f, and $\mathbf{\Pi} = \{\pi_1, \pi_2\}$ are weights for each of the keyword frequency vectors. The JS measure is symmetric and given the probabilities p_i and p_j of stories s_i and s_j, respectively, the distance measure of *Information Loss (IL)* is defined as

$$IL(s_i, s_j) = (p_i + p_j) \times JS_{\Pi}(s_i, s_j), \tag{5.13}$$

where $\Pi = \{\frac{p_i}{p_i+p_j}, \frac{p_j}{p_i+p_j}\}$.

In order to assess the quality of clustering results based on the content of the resulting clusters, many researchers have relied on the *expected entropy* measure, which quantifies the amount of randomness that exists in the set of clusters [7, 5]. In our experiments we compute the representative keyword frequency vector of each metastory using Equation 5.1, i.e., as the sum of individual story vectors. Then, we evaluate the entropy of each metastory representative. Intuitively, lower quality clusters will have large entropy, which in turn means that the distribution of keywords therein approaches a uniform distribution. Conversely, good clusters will have frequency vectors whose entropy is low, which in turn means that there is a clear separation of the set of keywords inside these clusters. For a set of metastories $\mathbf{M}$ obtained from some clustering algorithm and covering n stories we compute the expected entropy as

$$H(\mathbf{M}) = \sum_{m \in \mathbf{M}} p(m) H(f_m), \tag{5.14}$$

where $p(m) = |m|/n$ is the probability that m occurs within the set $\mathbf{M}$, and H is the entropy defined as above. Assuming that stories are uniformly distributed, the probability a metastory $p(m)$ is proportional to the number of stories within m.

The expected entropy of the clustering result depends on the number of clusters the algorithm produced. The entropy is minimum when all data points are left as individual clusters. Conversely, it is maximum when all data is clustered into one metastory. Therefore, we have to control this parameter in our experiment and ensure that algorithms are compared on a fair basis. Hence, we measure the entropies of clusterings resulting from all algorithms over a range of clustering sizes that includes individual stories at one end, and a single cluster containing all stories at the other. This involves systematically changing the algorithm parameters to affect the number of clusters produced. For aIB this is straightforward, while for ES and MCL this is achieved by filtering out weak edges and varying the algorithm sensitivity parameters.

Figure 5.2 shows the expected entropy each algorithm yields as a function of the aggregation ratio. The aggregation ratio is the amount of compression that is achieved via clustering each time. The lower this ratio, the

more clusters there are in our system and vice versa. The entropy values have been averaged over the seven story categories and the error bars show standard error. Small bars indicate that the entropy results are stable across all categories.

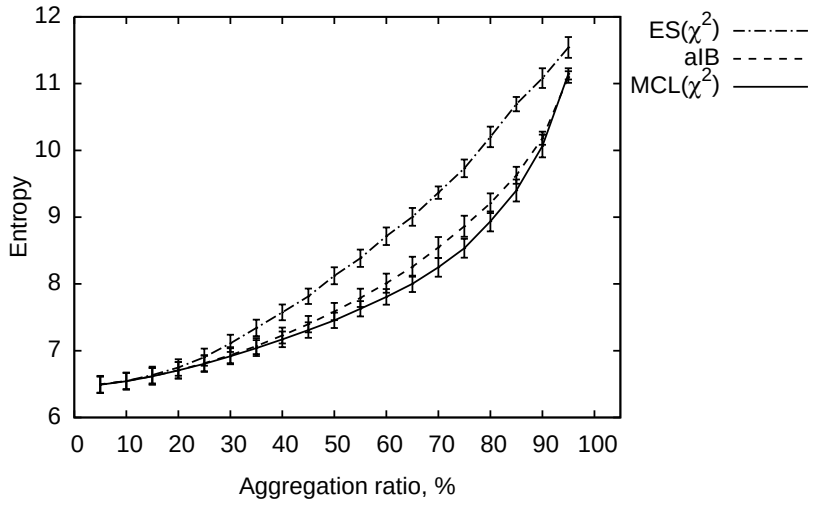


Figure 5.2: Clustering entropy as a function of aggregation ratio

The results of this experiment, as shown in Figure 5.2, provide us with a principled way to choose the best clustering method among those studied. For low aggregation ratios all three algorithms are equivalent, which is reasonable since we expect that fewer of the merges leading to metastories occur between strongly related stories, and these merges should happen first for any reasonable method. Beyond a 40% aggregation ratio the curves diverge with aIB showing a better performance. We note that beyond 12% of aggregation the ES algorithm constructs clusters that are characterized by larger entropy values. In the context of our particular problem we have observed that aggregation ratios over 30% tend to produce metastories that are too general and do not correspond to a well defined world event. We will, therefore, set the aggregation ratio to a fixed value of 20% for the evaluation of ranking results. At this level of aggregation, we could use either aIB or MCL. Since aIB provides a much finer control of the level of aggregation, we will use aIB as the clustering method in all the ranking evaluation tests.

5.4.3 Quantitative Comparison of Rankings

Having selected the clustering algorithm to produce the metastories, we turn to the analysis of the ranking results. Recall that we rely on the particular properties of symmetric similarity matrices to determine the ranks of

metastories. Thus, the ranking depends directly on how well the similarity measure captures the relationships between metastories. We use the following symmetric measures:

- The Jensen-Shannon JS distance measure as defined in Equation 5.12 with $\Pi = \{1/2, 1/2\}$.
- The χ^2 distance measure as defined in Equation 5.11.
- The *minimum (MIN)* and *maximum (MAX)* overlap between stories. MIN (resppectively MAX) is defined as the number of keywords two stories overlap, over the length of the smaller (respectively larger) of the two.
- The *Jaccard Coefficient, (JC)*, which measures the amount of keyword overlap between two stories as a fraction of the size of the union of their keywords.
- The *Cosine similarity, (COS)*, which measures the angle between two vectors of keyword frequencies.

All these measures (we refer the reader to [6] for their formal definition) have shown good performance in clustering and retrieval tasks. We selected JS because it is closely related to the IL measure used by aIB, but does not allow the size of the metastories to bias the similarity computation. Both χ^2 and JS are distance measures. In order to obtain consistent results, we convert them to similarity measures. Hence, given the distance $d(m_i, m_j)$ between two metastories, m_i and m_j, the similarity is computed by first normalizing the distance value $\hat{d}(m_i, m_j) = d(m_i, m_j)/max_{k,l}(d(m_k, m_l))$ and then subtracting the normalized distance value from 1. We construct the representative metastory vectors using $\mathbf{Sum}$ as in Equation 5.1 and then evaluate the ranking results produced by these different similarity measures.

We compare the rankings produced by all measures quantifying the differences using the normalized Kendall's tau distance, K. Intuitively, K is equal to the number of exchanges needed to convert one ranked list into another using the bubble sort algorithm [20]. For two ranked lists τ_1, τ_2 containing integers in the range $[1, \ldots, p]$ that indicate the ranks of each of the p metastories, the Kendall's tau distance is defined as $K(\tau_1, \tau_2) = \frac{2}{p(p-1)} \sum_{i,j} \hat{K}_{i,j}(\tau_1, \tau_2)$ where

$$\hat{K}_{i,j}(\tau_1, \tau_2) = \begin{cases} 0, & \text{if i,j are in the same order in both lists,} \\ 1, & \text{otherwise.} \end{cases} \tag{5.15}$$

Table 5.5 summarizes the pairwise distances between rankings obtained using each of the similarity measures. As suggested in the previous experiment, we used aIB and 20% aggregation ratio to obtain the metastories. The query lifespans were fixed to cover 100% of the time domain. Data from all the seven categories was used and, therefore, the values in the table correspond to average Kendall's tau distance computed over all the

datasets. The smallest and the largest distances are marked in bold. Observe that the only two measures that do not yield significantly different results are χ^2 and JS. The biggest difference, 0.326, is between Jaccard Coefficient and MAX measures.

It is worth to note here that a Kendall's tau distance of 0.5 indicates completely unrelated rankings, e.g. two random rankings, whereas the distance of 1 indicates inverse rankings. The bigger the random lists, the less likely it is that heir distance will deviate from 0.5. Since in our experiment we have about 700 titles, the probability of observing distance 0.45 between two rankings generated by random draws from uniform distribution is less than 1%. The above distance 0.326 between Jaccard Coefficient and MAX measure, therefore, indicates that there is significant relationship between the results and that all the measures are consistent about the ranking at large. Our aim with further experiments is to find out which of the measures provides the best result overall.

Table 5.5: Pairwise Kendall's tau distance between rankings obtained using different similarity measures

	χ^2	JS	JC	COS	MIN	MAX
χ^2	0	**0.013**	0.201	0.130	0.182	0.187
JS	-	0	0.206	0.137	0.173	0.182
JC	-	-	0	0.100	0.290	**0.325**
COS	-	-	-	0	0.256	0.284
MIN	-	-	-	-	0	0.217
MAX	-	-	-	-	-	0

Next, we check whether the size of the query window influences the difference between rankings of different measures. To simplify the visualization, we take χ^2 as the baseline and compare all the other measures to it for a set of queries. The latter were artificially designed to span multiple regions of the time domain and have their lifespans covering from 10% (approximately 6 days) to 100% of the 56-day time domain for which we have data. Figure 5.3 depicts the average Kendall's tau distance between each pair of rankings as a function of query lifespan. It is apparent that the distance values do not depend on the query.

The above experiments clearly indicate that the similarity function influences the final ranking independently of the time interval specified by the query. Therefore, the similarity function must be chosen in a principled way to yield the best ranking result. We continue our experiments evaluating the similarity functions with respect to a subjective ground truth.

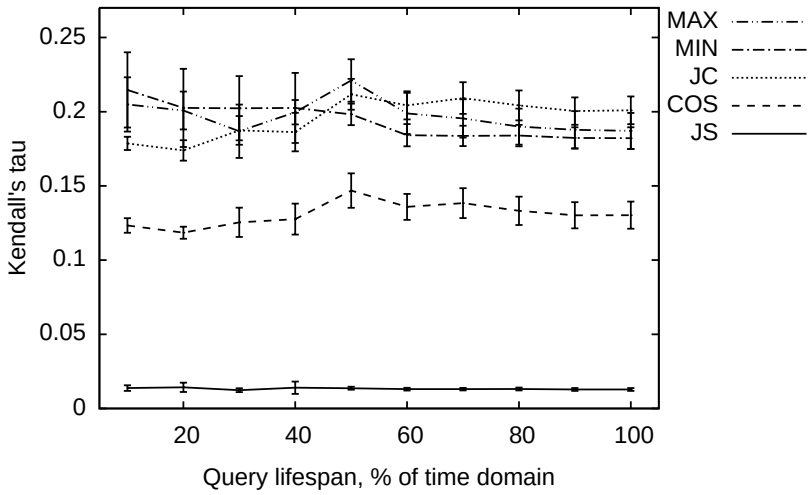


Figure 5.3: Kendall's tau distance between the rankings produced by χ^2 and other similarity measures as a function of query length

5.4.4 Subjective Evaluation of Ranking Quality

The quality of a given ranking is ultimately a subjective quantity as it depends on the particular interests of the user, the context in which they perform the query, and a multiplicity of other factors to which the ranking process has no access. To verify that the rankings produced by our framework are sensible we need to compare our results with the human perception of which metastories are more important. Although an optimal ranking is very hard to get, we will leverage two sources of opinion to estimate the way people would rank the metastories, and compare our results against them. First, we measure the impact of specific metastories in the social media as the number of blogs written about them. The assumption here is that more relevant stories will encourage people to write more blogs. Second, we obtain metastory ranking through the Amazon mechanical turk and use it as ground truth in our evaluation.

Blog Count as a Measure of Social Impact

As mentioned above, the platform offers access to social media sources, in particular to personal weblogs written by people who offer their opinion about developing stories. It is reasonable to assume that bloggers will spend more time writing about more relevant stories, and that in this way the relevance of metastories can be directly measured by looking at the number of blogs associated with them.

In this experiment we evaluate how well our similarity based ranking

corresponds to the social impact, i.e. the blog count, of each of the metastories. We generate queries as discussed in Section 5.4.3, and for each query we compute a ranking of metastories using our framework, and the corresponding ranking due to the number of blogs associated with each metastory. We then compute the Kendall's tau between these two.

Figure 5.4(a) shows the distance between our ranking and the ranking induced by measured social impact as a function of the size of the query window. Results are averaged over all categories. The results show that the best measure in terms of agreement with the social impact ranking is JS, followed closely by χ^2. The other measures lag behind in terms of agreement. This is reasonable and agrees with the conclusions from [47]. The average Kendall's tau distance for JS is 0.2914, and for χ^2 it is 0.2962, which is remarkable given that the ranking relies exclusively on the similarity scores with no access to social impact information. This verifies the validity of our ranking approach for situations in which the social impact factor is not available.

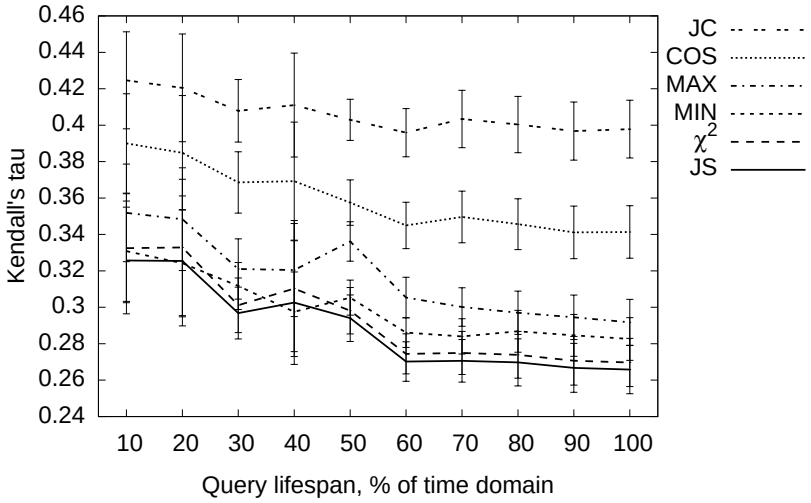
Figure 5.4(b) shows the results in terms of Kendall's tau distance from the *Blog Count* ranking when we consider each category separately. Overall, we observe that JS and χ^2 give the best ranking performance. It is worth noting that the differences in performance over different categories may be indicative of particular properties of stories in each category that may be exploited for ranking purposes. This remains a topic of future work.

Ranking Using Mechanical Turk

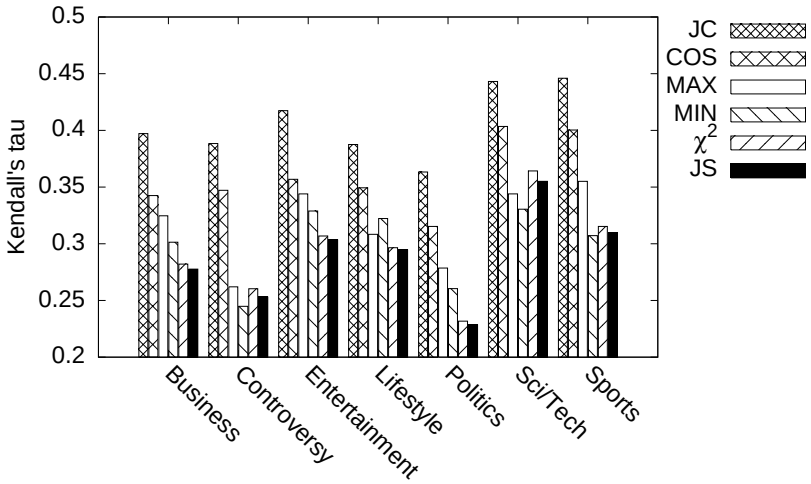
To further validate our framework we carried out an evaluation based on the Amazon Mechanical Turk[4] (AMT), a service that allows to collect user feedback. In our experiment we want the AMT users to elect the best ranking of metastories that we use as a ground truth to evaluate results of the framework. However, asking the AMT users to compare all possible rankings of metastories and choose the best one is not practical as there may be many metastories, and comparison of long lists is a very hard task. In such a situation the quality of an AMT experiment is likely to give poor results [4].

To overcome this problem we select a random and uniformly sampled subset of fifteen metastories per category. For each metastory we choose one representative title using the following procedure. First, we find the story whose keyword frequency distribution is closest to the representative distribution of the metastory in terms of the JS distance. Then, from titles of individual news and blog documents belonging to the story we choose the longest one. The selected story should be the most representative of the metastory, and the longest title offers the most information for the AMT user.

[4] `http://www.mturk.com`



(a)



(b)

Figure 5.4: Similarity-based ranking vs. social impact

Question:	Which story about (*category*) is more important?
Note:	Consider two story titles below. Choose the one that is more **important** in the category of stories about (*category*). Please choose according to IMPORTANCE and NOT your curiosity.
Option 1	*Title 1*
Option 2	*Title 2*

Figure 5.5: Human intelligence task template

Each AMT user is shown one pair of titles from the sampled list belonging to the same category. The user is asked to choose the title he/she considers more important. Each such question forms the so called Human Intelligence Task (HIT). Each HIT is made of a question, an explanatory note and the two metastory titles from which the user can only choose one to be ranked higher. The template of a HIT is provided in Figure 5.5. The field *category* is substituted by the corresponding category name of the pairs of titles we use. The fields *Title 1* and *Title 2* are replaced with respective titles.

The same HIT, i.e. the same pair of titles is shown to eleven different AMT users Intuitively, if a title gets more votes than any other title in its category, it should be ranked first. A ranked list of titles is constructed from the pairwise votes using the Schulze method [54]. This method satisfies the Condorcet criterion guaranteeing that the metastory title that is preferred over all the other candidates will win. The resulting list is then used as ground truth for comparison with the rankings produced by our framework and we refer to it as the *AMT list*.

We compare the rankings obtained using our framework and different similarity measures to the AMT list. Note that we only compare ranks of the metastories that are part of the AMT list. Table 5.6 shows the Kendall's tau between these pairs as well as between each distance measure and the social impact rank, measured by *Blog Count*, (BC), as described above. We observe that χ^2, MAX, and MIN give the best overall ranking with regards to the AMT. However, the standard error across different categories varies the least for χ^2 and JS indicating that these two measures are the most consistent. This is in agreement with the results reported previously for *Blog Count* in Figure 5.4(a), and is further verified by the comparison between the ranks induced by different distance measures and the *Blog Count* rank over the subset of stories sent to the AMT.

An interesting observation is that the distance between BC and the AMT list is larger than the distance between any distance measure and ei-

Table 5.6: Kendall's tau distance between rankings produced by different measures

	Business	Controversy	Entertainment	Lifestyle	Politics	Sci/Tech	Sports	Average	Std. Err.
χ^2 vs MTurk	0.407	0.371	0.486	0.371	0.429	**0.343**	0.467	**0.410**	0.019
JS vs MTurk	0.396	0.371	0.495	0.371	0.429	0.352	0.476	0.413	0.019
MAX vs MTurk	0.418	0.400	0.448	**0.248**	0.429	0.438	0.410	**0.398**	0.024
MIN vs MTurk	**0.363**	**0.362**	**0.429**	0.419	**0.314**	0.457	0.524	**0.410**	0.024
COS vs Turk	0.495	0.314	0.533	0.438	0.410	**0.343**	**0.400**	0.419	0.027
JC vs MTurk	0.473	0.371	0.543	0.429	0.448	0.362	0.410	0.433	0.022
BC vs MTurk	0.607	0.500	0.564	0.533	0.352	0.545	0.509	0.516	0.028
JS vs BC	**0.536**	0.303	**0.269**	0.178	0.363	**0.242**	0.236	**0.304**	0.041
MIN vs BC	0.607	0.212	**0.269**	0.311	**0.297**	0.348	**0.109**	0.308	0.054
χ^2 vs BC	**0.536**	0.333	0.282	0.178	0.363	0.258	0.236	0.312	0.041
COS vs BC	**0.536**	0.379	0.346	0.178	0.418	0.348	0.382	0.369	0.037
MAX vs BC	0.571	**0.182**	0.308	0.511	0.308	0.227	0.509	0.374	0.054
JC vs BC	0.679	0.424	0.449	**0.156**	0.396	0.379	0.455	0.419	0.054

ther BC or the AMT list. This could very well be caused by cultural and environmental differences between AMT users and the blogger community. However, it is encouraging to verify that in both cases the similarity measures do a good job of ranking metastories. To illustrate the kinds of stories ranked at the top by different methods, Table 5.7 shows titles for metastories in the politics category as ordered by the AMT, the χ^2 measure, and *Blog Count*.

The above experiments show that our ranking framework is feasible and produces good ranking results compared to rankings derived from human preferences as well as the social impact. Of the tested similarity measures, JS and χ^2 show the best ranking performance. It is also clear that the choice of similarity measure has a dramatic effect on ranking quality. This study provides strong evidence that within the context of story ranking, the use of aIB for clustering, together with either JS or χ^2 produces the best metastories, and the most accurate ranking.

5.4.5 HITS-Style Ranking vs PageRank

All the above experiments were conducted using the PageRank approach. However, HITS style ranking could be used as suggested in [37] by treating individual stories as authorities and metastories as hubs. A bipartite graph between stories and metastories is constructed with edges connecting stories and metastories. Edges are weighted by the similarity of the corresponding story and metastory representatives. The HITS algorithm takes as input the graph and returns a ranking of both, stories and metastories.

To test this, we quantify the Kendall's tau distance between the rankings obtained via the HITS algorithm, and those produced by PageRank as described above. The first two rows in Table 5.8 show the average Kendall's Tau distance for each similarity measure, and over all the categories. Also shown is the corresponding standard error. Observe that the rankings are very close independently of the similarity measure used. Next, we see how well HITS approximates the social impact ranking (blog count). As the data in the table shows, the average Kendall's tau distance between HITS and blog count is very similar to that of PageRank and blog count. However, PageRank is slightly but consistently better. In conclusion, due to the lower computational cost and better ranking performance, we recommend PageRank as the ranking method of choice.

5.5 Summary

We have presented a framework for the ranking of evolving stories through meta-aggregation. Metastory ranks are obtained in two steps. First, related stories are clustered into metastories. We explored the issue of selecting an

Table 5.7: Different rankings of titles in the Politics category

	Mechanical Turk	χ^2 ($K = 0.429$)	Blog count ($K = 0.352$)
1	House panel begins push on financial overhaul	Obama defends himself...	An Islamic Murderer meets...
2	Italy convicts former CIA agents in rendition trial	South benefits from stimulus...	Italy convicts former CIA...
3	Israel: commandos seize Hezbollah-bound arms ship	Chavez warns of Colombia war	Israel: commandos seize...
4	Afghan ministry: NATO strike kills Afghan forces	Palestinian President Calls...	Afghan ministry: NATO strike...
5	Chavez warns of Colombia war	Afghan ministry: NATO strike...	Chavez warns of Colombia war
6	Iran, North Korea nuclear disputes top Hillary Rodham Clinton's agenda on trip to Europe, Asia	Italy convicts former CIA...	Obama defends himself...
7	Palestinian President Calls For January Elections	Israel: commandos seize...	Palestinian President Calls...
8	Judge puts restraining order on the abortion law	Typhoon Slams Into Japan	South benefits from stimulus...
9	Obama defends himself against New Orleans critics	House panel begins push on...	HHS' Sebelius: Ample Flu...
10	HHS' Sebelius: Ample Flu Vaccine Will Be Available	HHS' Sebelius: Ample Flu...	Taos hotel owner tells...
11	An Islamic Murderer meets "Political Correctness"	An Islamic Murderer meets...	House panel begins push...
12	Typhoon Slams Into Japan	Greyhound will continue...	Judge puts restraining...
13	South benefits from stimulus construction boost	Judge puts restraining...	Iran, North Korea nuclear...
14	Greyhound will continue bus service in Manitoba, along with subsidies	Iran, North Korea nuclear...	Greyhound will continue...
15	Taos hotel owner tells Hispanic workers to change names	Taos hotel owner tells...	Typhoon Slams Into Japan

Table 5.8: Kendall's tau distance between HITS and PageRank approaches

	JS	χ^2	JC	COS	MIN	MAX
HITS vs PageRank						
Average	0.0257	0.0271	0.0520	0.0457	0.0366	0.0208
Std.err.	0.0097	0.0102	0.0197	0.0173	0.0139	0.0078
HITS vs Blog Count						
Average	0.2940	0.2984	0.4256	0.3697	0.3093	0.3047
Std.err.	0.1111	0.1128	0.1609	0.1397	0.1169	0.1152
PageRank vs Blog Count						
Average	0.2914	0.2962	0.4150	0.3616	0.2953	0.3096
Std.err.	0.1101	0.1119	0.1568	0.1367	0.1116	0.1170

optimal clustering method for this task, and showed that the MCL algorithm yields the best results. Second, implicit links are constructed based on metastory content similarity. We showed that the Jensen-Shannon distance measure provides the best results, and discussed the efficient computation of metastory ranks. We evaluated our framework in terms of the ground truth provided by the amount of weblogs written about a given metastory, and also in subjective terms through the use of the Amazon Mechanical Turk. Our results confirm the validity of the framework and its usefulness to determine a time-dependent rank for metastories in a completely unsupervised manner. Future work will focus on scalability issues, including the efficient computation of metastory representatives, and the incremental computation of ranking results.

CHAPTER 6

Conclusions

In this final chapter we summarize the issues discussed in the thesis and outline the contributions made. Next, we elaborate on the future research directions. Those include improvements to the temporal aggregation algorithms as well as extensions to the temporal ranking framework.

6.1 Summary

The work presented in this thesis emerged from the observation that most of the data stored in warehouses today is equipped with temporal dimension, yet existing tools for summarizing this kind of data are of little use as they do not meet the requirements of large data repositories. First, state of the art temporal aggregation operators do not allow explicit control over the result size. In case of instant temporal aggregation, that may lead a result larger than the argument relation. In case of span temporal aggregation the aggregation result may eventually hide important changes present in the underlying data. Such behavior is contrary to the very concept of temporal aggregation. Second, unlike traditional aggregation, grouping in temporal aggregation is unlikely to be static and known in advance. Instead, we showed in this thesis that the groups are dynamic and evolve through time.

Consequently, we argued that a good temporal aggregation operator will allow the user to specify explicitly the size of the output and the operator will satisfy it through such an approximation that introduces minimal amount of error into the aggregation result. Moreover, a technique to discover the aggregation groups from the data and disseminate the most important ones pertaining to any given time period is necessary.

We addressed the problem in two steps. First, we introduced a novel temporal aggregation concept, termed parsimonious temporal aggregation (PTA). It leverages from the previous temporal aggregation approaches, namely instant (ITA) and span (STA) temporal aggregation. PTA allows the user to specify a size- or an error- bound for the approximation. It first computes the ITA result and then merges similar adjacent tuples until the bound is satisfied. This way the operator starts from the most precise aggregation result, i.e. the ITA, and controls the size of the final output through data adaptive approximation.

Together with the definition of the new operator we researched the algorithms that could be used to evaluate PTA queries. We proposed two evaluation scenarios, offline and online, and suggested an evaluation strategy for each of them. Offline evaluation should be chosen when resources are not limited, computation time is not restricted, and the smallest approximation quality is desired. For such a scenario we proposed a novel dynamic programming algorithm that will find the best possible approximation of the instant temporal aggregation result. In the worst case scenario, for an argument relation of n tuples the algorithm takes $O(n^2c)$ time to produce an output relation of c tuples. It needs $O(n^2)$ space for computation. We showed that making use of temporal gaps in the data and aggregation groups in the query leads to linear computation time observed using real-world datasets.

The online evaluation scenario of the PTA queries would be chosen by

applications that require a quick response yet are willing to sacrifice precision. For example a data analyst may run a series of online queries and analyze preliminary results in order to pinpoint the exact query she is interested in before she engages in a more time consuming offline computation. For this scenario we proposed a greedy evaluation strategy. This strategy operates on an ITA result relation by choosing iteratively the most similar pairs of tuples and merging them until the size or error bound is satisfied. Such a strategy may, and is very likely to, introduce more approximation error than the dynamic programming approach would. Interestingly, we showed that the error ratio of greedy to optimal result is upper bounded by $O(\log n)$, i.e. the more data there is to process, the higher the ratio may be.

We introduced two novel greedy evaluation algorithms, for size- and error-bounded queries respectively. The distinctive feature of both algorithms is that they do not have to read the whole ITA relation before starting to merge tuples. Thus they combine the evaluation of ITA and merging into one process thereby saving computation space and dramatically reducing evaluation time. Most importantly, such early merging does not influence the final result, that is, they produce the same, or practically identical, output that the straight-forward implementation of the greedy strategy would. The algorithms run in $O(n \log(c + \beta))$ time and $O(c + \beta)$ space where β is typically very small, $\beta \ll n$, yet in the worst case it may become $c + \beta = n$.

Experimental evaluation of PTA algorithms revealed many interesting features of the new operator. First of all, we saw that temporal aggregation results often carry much redundant data and PTA discards it effectively. Thus PTA reduces aggregation results significantly yet introduces only small errors. Next, the evaluation of greedy algorithms revealed that the algorithms scale very well for huge datasets and the additional error they introduce is very small. In fact, they introduce the smallest additional error among all the non-optimal approximation methods we evaluated. Therefore, we would also like to recommend the use of the greedy algorithms also in other fields where compact representations have to be constructed, e.g. time series approximation.

Having succeeded in approximating the temporal aggregation we turned to the second part of the problem, that is, mining of aggregation groups from the data. We focused our research on a news and social media aggregation application where the need for aggregation group mining was evident. The application was able to cluster news, blogs and tweets, all related to the same story. We discovered, however, that the stories tend to be short in time and highly consistent, the long-running stories are usually split apart. Thus, in order to obtain a good overview of the data, aggregation was not enough, but the discovery of the most important and possibly long-running stories, i.e. aggregation groups, was the key.

We proposed a temporal ranking framework that consists of two steps. First, an offline clustering step identifies related stories and puts them into

one aggregation group, so called metastory. Second, online ranking queries can be answered. The metastories that overlap the time interval specified in the query are selected and ranked. For ranking we suggested an approach similar to that of PageRank. We construct implicit links between metastories based on their content similarity. We showed that as long as the similarity measure is symmetric the power method computation of rank vector is not necessary. Instead the rank of a metastory is computed as its total similarity to all the other metastories.

We performed extensive evaluation of the new framework. Our aim was to identify the best clustering algorithm for the metastory discovery and the best similarity measure for ranking. Among the different clustering algorithms that we tried, Markov clustering (MCL) showed the best results. Interestingly, we observed that about 20% of stories belong to long running meta-stories, and clustering beyond this threshold provides meta-stories that are overly general.

In the evaluation of the ranking step we wanted to determine which similarity measure provides the best ranking results and how well they reflect the public opinion. First of all, we observed that the choice of the similarity measure influenced the final outcome significantly. The same behavior could be observed for varying temporal queries leading us to the conclusion that the phenomena is due to the properties of the similarity measure, but not the underlying data or the lifespan of the query. Next, we used public opinion to evaluate the quality of rankings produced using our framework and different similarity measures. We obtained the public opinion in two ways. First, we measured blog count related to a given meta-story as an implicit user vote. Comparing the different rankings we found out that Jensen-Shannon divergence measure provides the best prediction of the public opinion expressed through blog count. In order to verify the latter finding, we employed Amazon Mechanical Turk service. In this case as well, the rankings produced by the framework were able to predict the public opinion to a large extent.

Summing up, the combination of the opinions obtained from the social media and the proposed ranking framework enables objective look over the historical perspective. The solution we proposed may help one to find out what the people of the world were talking about and what they considered the most important at any given historical time period. The technology, therefore, may impact not only the field of news aggregation but also the way the historical events are studied and their veracity is confirmed.

6.2 Future Work

The work discussed in this thesis forms the basis for a system that can aggregate and rank vast amounts of historical data. The proposed ap-

proaches can be further improved and extended in various directions. Here we outline the future research directions we find would be the most valuable.

First, recall that the parsimonious temporal aggregation operator is defined on top of instant temporal aggregation. However, cumulative (also termed moving-window) temporal aggregation is often used in commercial systems in parallel with ITA. As the operator suffers from the same drawbacks as ITA, it would be worthwhile to reduce its output using the techniques proposed in our work. In ITA aggregation result the time intervals that bear redundant data have more or less constant value at each time instant. In case of cumulative aggregation, the same time interval will show linear growth of the data values at each time instant. Therefore, a careful investigation of different error measures is worthwhile.

Second, there is a need to extend the greedy PTA algorithms to handle streaming temporal data and, consequently, operate in a distributed online environment. Large-scale internet crawlers discover vast amounts of new information every hour. They aggregate this data in parallel threads each taking care of its own separate aggregation group. Combining old and new data, sorting, and only then passing it to the PTA operator may be overly expensive. Instead, the PTA operator, at least the greedy evaluation algorithms, should be able to deal with each aggregation group separately and assume that the data comes sorted only along the time dimension, but not along the aggregation groups.

Third, the PTA operator may potentially be seen in a more general context of approximation of step functions by step functions with coarser granularity. In our future work we will investigate whether the underlying algorithm can be used to obtain improved results in histogram construction, time series indexing or detection of copy number variation which is a prominent problem in bioinformatics.

Regarding the error-bounded greedy PTA algorithm, we will work on estimating the maximal error that the reduction of an ITA result may introduce. We believe that in order to obtain a good estimate random time periods must be taken and corresponding aggregates computed. Instead, current sampling techniques usually select random data records.

Regarding the temporal ranking framework, our future work will focus on its scalability. First we want to reduce the number of keywords that are stored with each story leaving only the most indicative ones. Currently stories are often represented with tens of thousands of keywords. Such reduction may, however, affect the quality of metastory ranking as it is dependent on the similarity of keyword vectors. Since it is hard to speculate on wether the change would be positive or negative, we have to conduct a thorough experimental study of this issue.

Second, recall that for each temporal ranking query we have to recompute the representative vector of each metastory that falls into the time

period specified in the query. We think that this may be avoided with the help of cumulative temporal aggregation. We would store multiple representations of the same metastory that would correspond to different, ever increasing, time intervals. Again, it may appear that adjacent representations are very similar and can be merged using a technique similar to PTA as discussed previously.

Finally, we may be able to speed up the computation of the ranking result for any given query. Recall, that apart from recomputing the metastory representatives, we must also construct their pairwise similarity matrix. It may be possible to avoid such an expensive operation with the help of previously computed queries. It seems likely that many user queries, especially those spanning longer time periods and concerning older data, will cover the same or almost the same time periods. Thus it seems natural to reuse the answers of previous queries. Techniques such as rank aggregation may be used to combine the results of multiple queries that overlap the time period under consideration and obtain the answer for a given, previously unseen query. However, first of all the veracity of the assumption must be verified via a thorough user study.

In summary, in the future the temporal ranking framework should be optimized for better performance. Such optimization may lead to improved or deteriorated results. The experimental results provided in this thesis may serve as a baseline allowing one to quantify the amount of change.

Bibliography

[1] Gediminas Adomavicius and Jesse Bockstedt. C-TREND: Temporal Cluster Graphs for Identifying and Visualizing Trends in Multiattribute Transactional Data. *IEEE Trans. Knowl. Data Eng.*, 20(6):721–735, 2008.

[2] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient Search in Sequence Databases. In *Proceedings of the* 4^{th} *Conference on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[3] James Allan, Jaime Carbonell, George Doddington, Jonathan Yamron, Yiming Yang, and Umass Amherst. Topic detection and tracking pilot study: Final report, 1998.

[4] Omar Alonso, Daniel E. Rose, and Benjamin Stewart. Crowdsourcing for relevance evaluation. *SIGIR Forum*, 42(2):9–15, 2008.

[5] Periklis Andritsos, Panayiotis Tsaparas, Renée J. Miller, and Kenneth C. Sevcik. LIMBO: Scalable Clustering of Categorical Data. In *EDBT*, pages 123–146, 2004.

[6] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley-Longman, 1999.

[7] Daniel Barbará, Yi Li, and Julia Couto. Coolcat: an entropy-based algorithm for categorical clustering. In *CIKM*, pages 582–589, 2002.

[8] Klaus Berberich, Srikanta J. Bedathur, Thomas Neumann, and Gerhard Weikum. A time machine for text search. In *SIGIR*, pages 519–526, 2007.

[9] Pavel Berkhin. A Survey on PageRank Computing. *Internet Mathematics*, 2(1), 2005.

[10] Clive Best, Bruno Pouliquen, Ralf Steinberger, Erik Van der Goot, Ken Blackler, Flavio Fuart, Tamara Oellinger, and Camelia Ignat. Towards automatic event tracking. In *ISI*, pages 26–34, 2006.

[11] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Int. Conf. on Database Theory*, pages 217–235, 1999.

[12] Michael H. Böhlen, Johann Gamper, and Christian S. Jensen. Multidimensional aggregation for temporal data. In *EDBT*, pages 257–275, 2006.

[13] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[14] Pouliquen Bruno, Steinberger Ralf, and Deguernel Olivier. Story tracking: linking similar news over time and across languages. In *Proceedings of the 2nd workshop on Multi-source Multilingual Information Extraction and Summarization (MMIES'2008) held at CoLing'2008*, 2008.

[15] Kaushik Chakrabarti, Eamonn Keogh, Sharad Mehrotra, and Michael Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228, 2002.

[16] Chakra Chennubhotla. *Spectral Methods for Multi-Scale Feature Extraction and Data Clustering*. PhD thesis, University of Toronto, Dept. of Computer Science, 2004.

[17] Bertrand Clarke, Ernest Fokoue, and Hao Helen Zhang. *Principles and Theory for Data Mining and Machine Learning (Springer Series in Statistics)*. Springer, 2009.

[18] Hazem Elmeleegy, Ahmed K. Elmagarmid, Emmanuel Cecchet, Walid G. Aref, and Willy Zwaenepoel. Online piece-wise linear approximation of numerical streams with precision guarantees. *PVLDB*, 2(1):145–156, 2009.

[19] A.J. Enright, S. Van Dongen, and Ouzounis C.A. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Research*, 30(7):1575–1584, 2002.

[20] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. *SIAM J. Discrete Math.*, 17(1):134–160, 2003.

[21] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[22] Juozas Gordevicius, Johann Gamper, and Michael H. Böhlen. A Greedy Approach Towards Parsimonious Temporal Aggregation. In *15th International Symposium on Temporal Representation and Reasoning, TIME*, pages 88–92, 2008.

[23] Juozas Gordevicius, Johann Gamper, and Michael H. Böhlen. Parsimonious Temporal Aggregation. In *EDBT*, pages 1006–1017, 2009.

[24] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.

[25] K. Hasselmann, M. Latif, G. Hooss, C. Azar, O. Edenhofer, C.C. Jaeger, O.M. Johannessen, C. Kemfert, M. Welp, and A. Wokaun. The challenge of long-term climate change. *Science*, 302(5652):1923, 2003.

[26] Yannis E. Ioannidis. The history of histograms (abridged). In *VLDB*, pages 19–30, 2003.

[27] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *VLDB'98*, pages 275–286, 1998.

[28] 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 - february 1998 version. In *Temporal Databases, Dagstuhl*, pages 367–405, 1997.

[29] Panagiotis Karras. Optimality and scalability in lattice histogram construction. *PVLDB*, 2(1):670–681, 2009.

[30] E. Keogh, X. Xi, L. Wei, and C.A. Ratanamahatana. The UCR time series classification/clustering repository: www.cs.ucr.edu/$\sim$ eamonn/time_series_data/. Accessed on april 15, 2009.

[31] Eamonn J. Keogh and Michael J. Pazzani. A simple dimensionality reduction technique for fast similarity search in large time series

databases. In *In 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD*, pages 122–133. Springer, 2000.

[32] Jon Kleinberg. Temporal dynamics of on-line information streams. In *Data Stream Management: Processing High-Speed Data Streams*. Springer, 2004.

[33] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[34] Nick Kline and Richard T. Snodgrass. Computing temporal aggregates. In *ICDE Proceedings*, pages pp. 222–231, March 1995.

[35] Apostolos Kritikopoulos, Martha Sideri, and Iraklis Varlamis. BlogRank: Ranking weblogs based on connectivity and similarity features. In *Proceedings of the 2nd International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications, (AAA-IDEA)*, page 8, 2006.

[36] Oren Kurland and Lillian Lee. PageRank without hyperlinks: Structural re-ranking using links induced by language models. *CoRR*, abs/cs/0601045, 2006.

[37] Oren Kurland and Lillian Lee. Respect my authority!: HITS without hyperlinks, utilizing cluster-based language models. In *SIGIR*, pages 83–90, 2006.

[38] Amy Nicole Langville and Carl Dean Meyer. Survey: Deeper inside pagerank. *Internet Mathematics*, 1(3), 2003.

[39] Jure Leskovec, Lars Backstrom, and Jon M. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *KDD*, pages 497–506, 2009.

[40] Chung-Sheng Li, Philip S. Yu, and Vittorio Castelli. HierarchyScan: a hierarchical similarity search algorithm for databases of long sequences. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 546–553, 1996.

[41] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15(2):107–144, 2007.

[42] Inés Fernando Vega López, Richard T. Snodgrass, and Bongki Moon. Spatiotemporal aggregate computation: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):pp. 271–286, 2005.

[43] Bongki Moon, Inés Fernando Vega López, and Vijaykumar Immanuel. Efficient algorithms for large-scale temporal aggregation. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):pp. 744–759, 2003.

[44] Ramesh Nallapati, Ao Feng, Fuchun Peng, and James Allan. Event threading within news topics. In *CIKM*, pages 446–453, 2004.

[45] Shamkant B. Navathe and Rafi Ahmed. A temporal relational model and a query language. *Inf. Sci.*, 49(1-3):147–175, 1989.

[46] Jakub Piskorski, Hristo Tanev, Martin Atkinson, and Erik Van der Goot. Cluster-centric approach to news event extraction. In *New Trends in Multimedia and Network Information Systems*, pages 276–290. 2008.

[47] Yossi Rubner, Jan Puzicha, Carlo Tomasi, and Joachim M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. *Computer Vision and Image Understanding*, 84(1):25–43, 2001.

[48] Jin Shieh and Eamonn Keogh. iSAX: indexing and mining terabyte sized time series. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631, 2008.

[49] Noam Slonim and Naftali Tishby. Agglomerative information bottleneck. In *NIPS*, pages 617–623, 1999.

[50] Richard T. Snodgrass, Santiago Gomez, and L. Edwin McKenzie. Aggregates in the temporal query language TQuel. *IEEE Trans. Knowl. Data Eng.*, 5(5):826–842, 1993.

[51] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for computer graphics: a primer, part 1. *Computer Graphics and Applications, IEEE*, 15(3):76–84, May 1995.

[52] Abdullah Uz Tansel, James Clifford, and Shashi Gadia. *Temporal databases: theory, design, and implementation*. Benjamin-Cummings Publishing Co., Inc. Redwood City, CA, USA, 1993.

[53] Yufei Tao, Dimitris Papadias, and Christos Faloutsos. Approximate temporal aggregation. In *ICDE*, pages 190–201, 2004.

[54] Nicolaus Tideman. *Collective Decisions and Voting: The Potential for Public Choice*. Ashgate Publishing, 2006.

[55] Paul A. Tuma. *Implementing Historical Aggregates in TempIS*. PhD thesis, Wayne State University, Detroit, Michigan, 1992.

[56] Stijn van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, May 2000.

[57] Fushen Wang. Employee temporal data set. http://timecenter.cs.aau.dk/.

[58] Dietrich Wettschereck, David W. Aha, and Takao Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artif. Intell. Rev.*, 11(1-5):273–314, 1997.

[59] Gu Xu and Wei-Ying Ma. Building implicit links from content for forum search. In *SIGIR*, pages 300–307, 2006.

[60] Jun Yang and Jennifer Widom. Incremental computation and maintenance of temporal aggregates. *The VLDB Journal*, 12(3):262–s283, 2003.

[61] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary Lp norms. In *VLDB '00*, pages 385–394. Morgan Kaufmann Publishers Inc., 2000.

[62] Donghui Zhang, Alexander Markowetz, Vassilis J. Tsotras, Dimitrios Gunopulos, and Bernhard Seeger. Efficient computation of temporal aggregates with range predicates. In *PODS*, 2001.