Insights from database research, notably in the areas of data mining and similarity search, and advances in storage and microprocessor technology have enabled users to analyze and explore large-scale datasets. Data mining is the task of extracting previously unknown knowledge from data; similarity search encompasses techniques for finding objects similar by content. A prominent kind of data used in these tasks are temporal datasets, which stand out due to their information richness and their many possible applications. This thesis contributes novel, advanced methods for data mining and similarity search on temporal databases.

A major challenge in data mining research is the effectiveness of the approaches, corresponding to the quality of extracted patterns. The thesis addresses this challenge for the mining task of temporal clustering. First, a clustering technique is developed that is specifically designed for the requirements of real-world time series. Even in difficult settings with various measurement errors and misalignments between time series, it correctly identifies patterns concealed in temporal or dimensional subspaces of the data domain. Second, new methods for the complex task of mapping clusters between clusterings are contributed, for which two applications are investigated: tracing of evolving clusters in spatio-temporal data and the evaluation of clustering results in data stream scenarios.

The core of content-based similarity search systems and many data mining tasks are distance functions measuring the similarity between objects. An effective but also computationally expensive distance function for time series is based on adaptive warping on the time axis. This thesis introduces novel methods for queries under time warping. These methods exploit previously unused information in filter-and-refine frameworks for substantial runtime improvements. The anticipatory pruning technique utilizes distance information from a given filter step for rapid rejection of candidates in the refinement step, while the multiple query approach exploits shared characteristics between queries for joint pruning of candidates.

The presented approaches are experimentally analyzed and evaluated with respect to competing solutions. Overall, the techniques and results of this thesis represent a major advance in the research areas of data mining and similarity search on temporal data.
Mining and Similarity Search in Temporal Databases

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Hardy Kremer
aus Köln

Berichter: Universitätsprofessor Dr. rer. nat. Thomas Seidl
Privatdozent Dr. rer. nat. Peer Kröger

Tag der mündlichen Prüfung: 14.10.2013

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.
Hardy Kremer:

Mining and Similarity Search in Temporal Databases

1. Auflage, 2013

Ergebnisse aus der Informatik
Band 1/2013

Gedruckt auf holz- und säurefreiem Papier, 100% chlorfrei gebleicht.

Apprimus Verlag, Aachen, 2013
Wissenschaftsverlag des Instituts für Industriekommunikation und Fachmedien an der RWTH Aachen
Steinbachstr. 25, 52074 Aachen
Internet: www.apprimus-verlag.de, E-Mail: info@apprimus-verlag.de

Printed in Germany

ISBN 978-3-86359-157-1

D 82 (Diss. RWTH Aachen University, 2013)
# Contents

Abstract / Zusammenfassung 1

1 Overview 5
   1.1 Introduction .............................. 5
   1.2 Contributions and Thesis Structure .......... 9

I Effective Time Series Clustering 15

2 Subspace Clustering: Introduction 17

3 Subspace Mining in Time Series Databases 19
   3.1 Motivation .................................. 20
   3.2 Comparison with Related Work ............... 23
   3.3 Robust, Effective Clustering of Multivariate Time Series .... 26
   3.4 Efficient Computation ........................ 36
   3.5 Experiments .................................. 41
   3.6 Conclusion ................................... 46

II Effective Cluster Mapping for Temporal Data 47

4 An Evaluation Measure for Clustering on Data Streams 49
   4.1 Motivation .................................. 50
   4.2 Comparison with Related Work ............... 51
   4.3 Cluster Mapping Measure ........................ 53
   4.4 Analysis and Refinement ...................... 62
   4.5 Experiments .................................. 66
4.6 Stream Clustering Evaluation in MOA ............. 76
4.7 Conclusion .................................. 78

5 Tracing of Evolving Subspace Clusters 79
5.1 Motivation .................................. 80
5.2 Comparison with Related Work .................. 83
5.3 Model for Subspace Cluster Tracing ............. 85
5.4 Experiments .................................. 100
5.5 Application Scenario .......................... 104
5.6 Conclusion .................................. 105

III Introduction to Similarity Search 107

6 Distance-Based Similarity Search: Introduction 109
6.1 Distance Functions and Similarity Queries ............ 109
6.2 Efficient Query Processing ........................ 112

7 Subspace Clustering for Efficient Query Processing 117
7.1 Motivation .................................. 118
7.2 Comparison with Related Work .................... 121
7.3 Subspace-Clustering-Based Indexing ............... 122
7.4 Experiments .................................. 133
7.5 Conclusion .................................. 144

IV Efficient Time Series Similarity Search 145

8 Dynamic Time Warping (DTW) 147
8.1 Introduction to DTW .......................... 147
8.2 DTW Applications ............................ 153

9 Anticipating Distances for Efficient DTW Query Processing 157
9.1 Motivation .................................. 158
9.2 Comparison with Related Work .................. 159
9.3 Anticipatory DTW ............................ 160
Abstract

Insights from database research, notably in the areas of data mining and similarity search, and advances in storage and microprocessor technology have enabled users to analyze and explore large-scale datasets. Data mining is the task of extracting previously unknown knowledge from data; similarity search encompasses techniques for finding objects similar by content. A prominent kind of data used in these tasks are temporal datasets, which stand out due to their information richness and their many possible applications. This thesis contributes novel, advanced methods for data mining and similarity search on temporal databases.

A major challenge in data mining research is the effectiveness of the approaches, corresponding to the quality of extracted patterns. The thesis addresses this challenge for the mining task of temporal clustering. First, a clustering technique is developed that is specifically designed for the requirements of real world time series. Even in difficult settings with various measurement errors and misalignments between time series, it correctly identifies patterns concealed in temporal or dimensional subspaces of the data domain. Second, new methods for the complex task of mapping clusters between clusterings are contributed, for which two applications are investigated: tracing of evolving clusters in spatio-temporal data and the evaluation of clustering results in data stream scenarios.

The core of content-based similarity search systems and many data mining tasks are distance functions measuring the similarity between objects. An effective but also computationally expensive distance function for time series is based on adaptive warping on the time axis. This thesis introduces novel methods for queries under time warping. These methods exploit previously unused information in filter-and-refine frameworks for substantial runtime improvements. The anticipatory pruning technique utilizes distance information from a given filter step for rapid rejection of candidates in the refinement step, while the multiple query approach exploits shared characteristics between queries for joint pruning of candidates.

The presented approaches are experimentally analyzed and evaluated with respect to competing solutions. Overall, the techniques and results of this thesis represent a major advance in the research areas of data mining and similarity search on temporal data.
Zusammenfassung

Neue Erkenntnisse in der Datenbankforschung, insbesondere in den Bereichen des Data Mining und der Ähnlichkeitssuche, und die fortschreitende Entwicklung von Speichertechnologien und Mikroprozessoren ermöglichen die Analyse und Exploration von großen Datenmengen. Während es im Data Mining das Ziel ist, unbekanntes Wissen aus Daten zu extrahieren, behandelt die Ähnlichkeitssuche Techniken des inhaltsbasierten Objektvergleichs. Im Rahmen dieser Aufgaben erfreuen sich temporale Daten einer wachsenden Beliebtheit. Sie zeichnen sich durch ihren hohen Informationsgehalt und ihre zahlreichen Anwendungsmöglichkeiten aus. In dieser Dissertation werden neue Techniken des Data Mining und der Ähnlichkeitssuche für temporale Daten entwickelt.

Eine große Herausforderung in der Data-Mining-Forschung ist die Effektivität der vorgeschlagenen Verfahren, die sich in der Qualität der extrahierten Muster äußert. Im Bereich des Zeitreihen-Clustering wird daher eine neue Methode eingeführt, die sich an den Anforderungen von Realdatensätzen orientiert: Das Verfahren erkennt Muster, die in Teilräumen und Teilsequenzen verborgen sind, und ist in Hinblick auf Messfehler und zeitliche Verschiebungen sehr robust. Im Weiteren werden neue Techniken für die Aufgabe der Clusterabbildung entwickelt, bei der eine Zuordnung zwischen den Clustern zweier Clustermengen gesucht wird. Hierbei werden zwei Anwendungen betrachtet: Die Verfolgung von sich entwickelnden Clustern in raum-zeitlichen Daten und die Evaluierung von Clustering-Ergebnissen in Datenstrom-Szenarien.


Die in der Arbeit entwickelten Methoden werden experimentellen Analysen unterzogen und im Vergleich zu Konkurrenzverfahren evaluiert. Insgesamt stellen die vorgestellten Ansätze und Resultate einen wesentlichen Fortschritt in den Forschungsgebieten des Data Mining und der Ähnlichkeitssuche auf temporalen Daten dar.
Chapter 1

Overview

1.1 Introduction

Ongoing advances in data storage technology, processing power, and database research have enabled users to explore and automatically analyze large-scale datasets, whose sheer sizes rendered manual human analysis nearly impossible. Database research deals with several types of data. From these types, temporal data has become very popular due to its information richness that facilitates many interesting applications. Based on knowledge contained in temporal data, behavior of entities and systems can be compared, categorized, or even explained. Diseases may be detected at an early stage, frauds and system failures may be prevented. Decision makers can use temporal knowledge to predict future trends, as for example sales figures and stock market developments.

Temporal data occurs in many domains and in various representations. Examples include stock-trading data, multimedia content as audio and video, moving object trajectories, interaction sequences, real-time magnetic resonance imaging, and the output of complex simulations in science and engineering. Temporal data can be used to represent the history of observed entities and systems over time, resulting in network-traffic logs, heart rate values in biometrics, or engine readings in the automotive industry. Figure 1.1 shows illustrations for some of the mentioned temporal data.
Without effective processing techniques, users are not able to explore and analyze large-scale datasets, and this is especially true for temporal data. Accordingly, automatic techniques were introduced to mine databases for relevant knowledge. In the presence of vast amounts of data, these mining techniques are embedded in a general processing framework, known as Knowledge Discovery in Databases (KDD) [HK01].

The KDD process is a sequence of several steps, as illustrated in Figure 1.2. Starting with raw data originating from a single or several sources, noise and inconsistencies are removed and missing values are addressed. The cleansed data is then integrated into a single database, known as data warehouse. This unified view allows users to select the relevant parts of the data, in which they suspect hidden knowledge. After selection, the data is transformed to meet the requirements of the subsequent Data Mining step, which is the KDD process’ main component: the applied data mining technique extracts hidden patterns from the data. Dependent on the analyzed data and the data mining task, patterns can be, for example, models or groupings of objects. In the final step, evaluation helps in selecting interesting patterns that are presented to the user. Presentation is most often done by using visualization techniques.

In this thesis, we focus on the data mining step of the KDD process, or more concretely, the task of cluster analysis. Cluster analysis, also known as clustering, is the automatic grouping of similar objects while separating dissimilar ones, without using prior knowledge about the data. The found
1.1. Introduction

Figure 1.2: The process of Knowledge Discovery in Databases (KDD).

clusters represent the hidden structure of the analyzed database, corresponding to newly obtained knowledge. Ensuring that found clusters are accurate reflections of the data’s actual patterns is the major challenge in clustering research and still far from solved. In this thesis, we develop novel effective methods for clustering on temporal data. In particular, we contribute techniques for extracting high-quality clusters and techniques for detecting connections between given cluster sets.

The core of many mining techniques are vast amounts of similarity estimates between the objects of the analyzed database; for example, in cluster analysis, similar objects are assigned to the same cluster. Similarity Search, also known as query by content [Fal96], is the task of searching large databases for objects that are similar to a given query object. The similarity estimates between compared objects are based on their actual content, such as the values of time series. A similarity query is illustrated in Figure 1.3, where the objective is to find the most similar time series from the given database. Many effective distance measures for time series have been proposed, as for example the prominent Dynamic Time Warping distance illustrated in Figure 1.4. Similarity search can, dependent on literature and history of research, be seen as a data mining task embedded in the KDD process [Mör06]. If a distance measure for a specific object type is defined, standard data mining methods can be directly applied, as for example k-medoids for clustering and the k-nearest-neighbor-algorithm for classification.

A difference between similarity-based data mining and stand-alone similarity search is that the latter raises only small numbers of single similarity queries. However, in both data mining and similarity search, fast processing
Figure 1.3: Illustration of time series similarity search. For the given query, the most similar time series in the database is marked in gray. This specific task is also known as nearest neighbor search.

of similarity queries is essential for rapid exploration and mining of the analyzed data. In this thesis, we propose novel techniques for fast processing of time series similarity queries under the computationally expensive dynamic time warping distance.

Three kinds of temporal data are used throughout this thesis. *Time series* are the most frequent form of temporal data. They correspond to sequences of data points ordered by a function of time. The data points are typically measured at successive, uniformly-spaced points in time. Per point in time, a time series consists of either a single observation or of several ones; in the former case, time series are termed univariate or one-dimensional, in the latter case, there are called multivariate or multidimensional. Multiple variables allow the characterization of complex behaviors by time series. Figure 1.5(a) shows an example for a two-dimensional time series. *Spatio-temporal data* contains both space and time information. It can be categorized into spatio-temporal data of moving objects and spatio-temporal grid data. The former are usually location sequences in the Euclidean space,

Figure 1.4: Example of the Dynamic Time Warping distance in a two-dimensional space. The compared time series are marked in red and blue. The time warping is illustrated by black lines.
which can be enriched by additional attributes. Trajectories, i.e. time series where locations are captured at irregular points in time, belong to this category. Spatio-temporal grid data is based on a spatial grid for several consecutive time steps. Examples are satellite imagery, medical imagery like real-time MRI, and the output of complex simulation models. Dependent on the number of attribute and grid resolution, such datasets can be very large and high-dimensional. Figure 1.5(b) shows simulated oceanographic grid data spanning a period of four months. Data streams are sequences of data items that need to be processed as they arrive. They correspond to an possible infinite sequence, and in the course of time, dynamic changes in the data distribution are common. Figure 1.5(c) illustrates an infinite data stream.

1.2 Contributions and Thesis Structure

In this thesis, we introduce novel mining and similarity search techniques for different kinds of temporal data. The thesis is structured into four main parts and a summary. In Part I, we present a new method for clustering of multivariate time series, and in Part II we develop techniques for mapping clusters between clusterings obtained from data streams and spatio-temporal data. Part III gives an introduction to similarity search and its efficient processing, and in Part IV we contribute new techniques for speeding-up similarity search in time series databases with respect to a complex distance measure. In the following, we summarize their content and we take a peek at the specific challenges and research contributions.
Part I

In the first part, we develop a novel method for clustering of multivariate time series data, which is based on the principle of subspace clustering. Since subspace clustering is the basis of several techniques in this thesis (Chapters 3, 5, and 7), Chapter 2 gives a brief introduction to the topic with respect to non-temporal vector data.

In Chapter 3 we contribute a method called Robust Time Series Clustering (RTSC). Real world temporal data is often complex in the sense that interesting patterns are neither bound to the whole dimensional nor temporal extent of the data domain, and prone to aspects like measurement errors and misalignments between time series. Under these conditions, patterns mined by existing multivariate time series clustering and temporal subspace clustering techniques cannot correctly reflect the true patterns in the data; no new knowledge is gained for the user. RTSC tackles these challenges: Clusters are defined by sets of objects and individual sets of relevant intervals per dimension. RTSC handles misaligned time series by adaptively shifting them in the time domain, and it achieves robustness to measurement errors by allowing certain fractions of deviating values in each relevant point in time. These are major contributions, since existing approaches for temporal subspace clustering are very rigid: they rule out that a point in time is relevant in one dimension but not in another dimension, and they have no concept for flexibility on the time axis.

Part II

In the second part, we introduce new techniques for the task of mapping clusters between different clusterings; that is, given two sets of clusters, a mapping between the sets is determined such that clusters in the first set are matched to their most fitting counterparts in the second cluster set. We consider two different application scenarios for this complex task, and we develop an independent mapping method for each application.

In Chapter 4 we develop an evaluation measure for stream clustering results, called Cluster Mapping Measure (CMM). There is vast amount of
stream clustering algorithms in the literature, but hardly any attention has
been paid to measures that evaluate how accurately the obtained stream
clusterings reflect the data. This task is challenging, and measures developed
for non-temporal scenarios cannot reflect errors attributable to emerging,
splitting, or moving clusters correctly. The CMM indicates different types
of errors by taking the important properties of evolving data streams into
account. One of the main contributions is the novel mapping component
based on class distributions of clusters. Using these distributions, incorrect
cluster pairings caused by emerging and disappearing clusters are avoided
effectively.

A novel approach for tracing clusters in spatio-temporal data is presented
in Chapter 5. Generally, tracing algorithms are used to mine the tempo-
ral evolutions of clusters. The mapping components of existing tracing ap-
proaches rely on unique object identifiers; that is, clusters are traced by mea-
suring the fraction of objects two clusters have in common. This mapping
concept, however, cannot account for clusters with different object sets that
represent the same phenomena in the course of time. We contribute the
novel paradigm of object-value-based tracing, which maps clusters according
to their characteristics, independently of individual object identities. Since,
in complex data, clusters are often hidden in subspaces, we develop a con-
cept for comparing clusters manifested in different subspaces. Both contribu-
tions are implemented in the mapping component that uses a single distance
measure incorporating both subspace and value similarity.

Part III

In the third part, we give a general introduction to content-based similar-
ity search. In Chapter 6, we define different query types and prominent
concepts for speeding up query processing, namely filter-and-refine and in-
dexing structures.

In Chapter 7 we then develop an indexing structure for high-dimensional
complex data that serves as an example for both concepts and how they can
be tightly integrated. Most of the existing indexing structures exploit global
correlations between dimensions to reduce the dimensionality of the indexed space. In complex data, however, correlations are often locally constrained to a subset of the data and every object can participate in several correlations. Ignoring these different correlations and discarding the same set of dimensions for each object results in an information loss with direct impact on the achievable query performance. We tackle this challenge in our index by using local correlations to increase the information content for each single object in the index compared to a global approach. The index structure corresponds to a multi-representation of objects reflecting their multiple correlations; i.e., besides the general increase of information per object, we provide several individual representations for each single object. Combined, these concepts enable more effective pruning.

Part IV

In the fourth part, new methods for fast processing of time series similarity queries under a complex distance measure are investigated. First, Chapter 6 introduces Dynamic Time Warping (DTW), a highly effective but also computationally expensive distance measure for time series, which is used in many domains. We discuss the computation of DTW, and present two application scenarios in which efficient query processing under DTW is crucial: detection of suspicious patterns in streams and video copy detection.

In Chapter 9, we contribute a filter-and-refine DTW algorithm termed Anticipatory DTW. Existing filter-and-refine frameworks for DTW discard relevant information acquired in the filter step, and recompute this information in the refinement step. This recalculation of already existing information corresponds to a massive computational overhead. Our new approach exploits the previously unused information during the refinement, to obtain an accurate estimate of the actual DTW distance. This incurs hardly any overhead and enables a substantial faster rejection of false candidates. We characterize a class of applicable filters, comprising state-of-the-art lower bounding distances for DTW, and we prove that our approach is lossless for these filters.
In today’s applications, queries are often similar or even share subsequences, and they arrive in vast numbers. Existing algorithms for efficient DTW query processing, however, cannot process multiple DTW queries simultaneously, slowing down overall processing. In Chapter 10, we develop an efficient approach for this challenging task of processing multiple DTW queries at the same time. We formalize the multiple query extensions to range queries and $k$NN queries, and we even support range queries with different pruning thresholds. Our solution exploits common characteristics between queries by pruning database time series with respect to hierarchically nested query sets, and we prove a lower-bounding property that guarantees no false dismissals. Our technique can be flexibly combined with existing DTW lower bounds or other single DTW query speed-up techniques.

Part V

In the final part of the thesis, we summarize all contributions and give a short overview of promising future work.
Part I

Effective Time Series Clustering
Chapter 2

Subspace Clustering: Introduction

The concept of Subspace Clustering is the foundation of several techniques in this thesis (Chapters 3, 5, and 7). This chapter gives a short introduction to the topic with respect to non-temporal vector data.

Clustering is a mining task for automatic grouping of similar objects while separating dissimilar ones. For today’s high dimensional data, however, traditional fullspace clustering approaches fail to detect meaningful patterns since irrelevant dimensions obfuscate the clustering structure [HKK+10, BGRS99]. Using global dimensionality reduction techniques as Principle Components Analysis [Jol02] is no solution for this problem: by definition, all objects are projected to the same lower dimensional subspace. However, as Figure 2.1 illustrates, each cluster might have locally relevant dimensions and objects can be even part of multiple clusters in different subspaces. In the figure, the blue circles form a subspace cluster with the relevant dimensions 1 and 2, while the red squares depict a good grouping in the dimensions 3 and 4. The green triangle is an outlier, since it shows no similarity to other objects in any of the subspaces. Such effects cannot be captured in the full-dimensional space or by applying global dimensionality reduction.

Subspace clustering [APW+99, AGGR98, PJAM02, YM03, KKRW05] was introduced for non-temporal vector data, aiming at detecting locally relevant dimensions per cluster. They analyze arbitrary subspace projections of the data to detect the hidden clusters. Typical non-temporal applications for
subspace clustering include gene expression analysis, customer profiling, and sensor network analysis. In each of these scenarios, subsets of the objects (e.g., genes) are similar regarding subsets of the dimensions (e.g., different experimental conditions).

Recent research in subspace clustering has introduced several models and algorithms. Surveys can be found in [KKZ09, PHL04], and differences between the models are analyzed in [MZK+09, MGAS09]. A general classification of the models can be done by considering the possible overlap of clusters. Partitioning approaches [APW+99, BKKK04, PJAM02, YM03] force the clusters to represent disjoint object sets, while in non-partitioning approaches [AGGR98, KKK04, MAG+09] objects can belong to several clusters.

Subspace clustering is provided to the end user by data mining frameworks like ELKI [AGK+12], OpenSubspace [MGAS09] for WEKA [HFH+09], and our subspace clustering extension∗ for KNIME [BCD+07].

In this thesis, the paradigm of subspace clustering is the basis of several novel techniques. A method for time series clustering based on subspace clusters is introduced in Chapter 3. An approach for tracing evolving subspace clusters in spatio-temporal datasets is presented in chapter 5. Finally, in Chapter 7, we speed up similarity search query processing by applying the concept of subspace clustering in the construction phase of an hierarchical indexing structure.

∗Our subspace extension for KNIME has been published in the Workshop Proceedings of the 12th IEEE International Conference on Data Mining (ICDMW 2012)[GKM+12].
Chapter 3

Subspace Mining in Time Series Databases

∗ Mining temporal multivariate data by clustering is an important research topic. In today's complex data, interesting patterns are often neither bound to the whole dimensional nor temporal extent of the data domain. Under these conditions, patterns mined by existing multivariate time series clustering and temporal subspace clustering techniques cannot correctly reflect the true patterns in the data. Their effectiveness is further impeded by aspects unavoidable in real world data: misalignments between time series, for example caused by out-of-sync sensors, and measurement errors.

In this chapter, we propose a novel clustering method that mines temporal subspace clusters reflected by sets of objects and relevant intervals. Relevant intervals indicate those points in time in which the clustered time series show a high similarity. We enable flexible handling of misaligned time series by adaptively shifting time series in the time domain, and we achieve robustness to measurement errors by allowing certain fractions of deviating values in each relevant point in time. We show the effectiveness of our method in experiments on real and synthetic data.

∗This chapter has been published in the Proceedings of the 12th IEEE International Conference on Data Mining (ICDM 2012) [KGHS12a].
3.1 Motivation

Clustering of temporal data is a major area of data mining research [DSP05, Fu11, Lia05, WWW07, WY05]. Temporal data reflect the changing state of an observed system over time. Datasets are often multivariate, with each attribute representing a distinct aspect of the observed objects. Clustering approaches group time series based on their similarity. Clusters of time series correspond to groups of objects having a similar evolution over time, and detected clusters represent these evolutions.

In many applications existing approaches for clustering univariate or multivariate time series are ineffective due to a specific aspect of the analyzed data: Temporal patterns of interest often only exist over a partial temporal extent of analyzed time series, i.e. they are constrained to an interval, and a single multivariate temporal pattern can have different intervals for each of its dimensions, as it is illustrated in Figure 3.1. More concretely, time series belonging to one cluster only have similar values in these intervals, and values in the remaining intervals are noisy. Also, for some clusters there are dimensions in which there is no similarity between the time series. In the following, the intervals and dimensions belonging to a cluster are called relevant, while the remaining intervals and dimensions are called non-relevant. If non-relevant intervals are considered in deciding which time series are
3.1. Motivation

Figure 3.2: Spoken Arabic Digit data: plot of first dimension of all classes.

grouped together, clusters can result that do not reflect the true patterns in the data.

Our novel subspace clustering related approach handles this aspect by using an effective cluster model that distinguishes explicitly between relevant and non-relevant intervals for each cluster, and only relevant intervals are incorporated into the decision which time series belong to a specific cluster. Our approach prevents incoherent time series clusters, i.e. clusters that have points in time that do not belong to any of the clusters’ individual intervals. This prevents single (incoherent) cluster which would better be represented by several (coherent) clusters.

In real world data, aspects like misalignments between time series and deviating values are unavoidable. In the following, we explain why existing subspace clustering algorithms and the new approach described above are not sufficient in these scenarios.

Misalignments between time series that contain similar patterns are common in time series datasets. Reasons could be out-of-sync sensors, or that the same phenomena occurred slightly shifted in time. Figure 3.2 shows a real example, a plot of all classes of the Spoken Arabic Digit dataset from the UCI Machine Learning Repository [FA10]. (It only shows a sampled set of all time series.) In this plot, equal colors mark the same class. The plot shows that time series of some classes are out-of-sync to each other. An effective clustering algorithm has to be able to compensate for these misalignments and thus place the time series in the correct clusters. Generally, misalignments are well known in time series similarity research, where several distance measures were introduced to cope with this problem. Most of them
are based on the idea of finding a new alignment between the compared time series to reduce the dissimilarity of the corresponding values. Existing subspace clustering approaches, however, do not aim at comparing different dimensions. While this is reasonable for standard vector data, where each dimension represents a distinct aspect, it is an essential requirement for compensating misalignments between time series. In the worst case, subspace clustering approaches exclude time series from clusters because the misalignment obfuscates the similarity to the other time series contained in the clusters.

Measurement errors, or more generally, deviating values, are another aspect of real world data. Figure 3.3 shows the 11th dimension of a class of the Spoken Arabic Digit dataset. All time series show a similar behavior, but there are points in time in which some of them highly deviate. Subspace clustering approaches would either omit these time series from the cluster or they would consider the corresponding point in time as irrelevant, possible losing a pattern that discriminates the cluster from others. The reason is that subspace clustering assumes that dimensions are either relevant or irrelevant for a whole cluster, and there is no possibility that a relevant dimension of a cluster could contain some noise.

In this chapter, we propose a novel approach for clustering of multivariate time series. We introduce the paradigm of temporal coherent time series subspace clustering, in which each cluster has a set of time series and relevant intervals. We enable effective handling of misalignments by individually shifting of time series such that a better matching in a cluster’s relevant intervals is obtained. We achieve robustness with respect to deviating values by
allowing a certain fraction of deviating values per point in time in relevant
intervals. Since our problem is NP-hard, we propose an algorithm for our
model that delivers approximative clusterings of high quality.

Before we present our model, we discuss related work.

3.2 Comparison with Related Work

Our method is related to two clustering research areas: time series clustering
and subspace clustering. In the experiments, we compare our technique to
both areas.

**Time Series Clustering.** There is much research on clustering of uni-
variate time series data, and we recommend the surveys in [Fu11, Lia05].
Early work on multivariate time series clustering in [Oat99] uses clustering
to identify outliers. Multivariate clustering methods based on statistical fea-
tures were introduced in [DSP05, WWW07, WY05]. There are no concepts
in these approaches to discovery patterns hidden in parts of the dimensional
or temporal extents of time series.

Most clustering approaches are based on an underlying similarity measure
between time series. In the area of similarity search, there are noise-robust
similarity measures based on partial comparison and local shifting, as for ex-
ample the concept of the Longest Common Subsequence (LCSS) [VKG02].
LCSS allows non-matched elements and local stretching on the time axis
without rearrangement of the order of elements. The similarity of two time
series $x$ and $y$ is given by the length of the longest of all possible common
subsequences emerging after deleting arbitrary elements from $x$ and $y$. Fig-
ure 3.4 shows an example, in which mappings between points in time are
indicated by dotted lines. Overall, the maximal number of mappings with-
out intersection (i.e., if $(t_1, t_2)$ and $(t'_1, t'_2)$ are mapped and $t_1 < t'_1$, then $t_2 < t'_2$
has to hold) determines the similarity between both sequences. In the figure
we get a LCSS similarity of 11, i.e. there are 11 mappings. Intuitively, non-
mapped time points correspond to deleted values, and consequently deviat-
ing values are handled by excluding them from the measure. Misalignments
can be handled because for a mapping $(t_1, t_2)$ the setting $t_1 \neq t_2$ is possible.
In LCSS, mappings can be constrained to the temporal neighborhood. Thus, for a mapping \((t_1, t_2)\) as constraint \(|t_1 - t_2| \leq \text{offset}\) has to hold. This offset is the maximal temporal shift the user is willing to accept (e.g., due to domain knowledge). Measures like LCSS or Dynamic Time Warping (DTW) are called elastic. An introduction to DTW can be found in Chapter 6 of this thesis, and an overview of elastic measures is given in [DTS+08]. From the different elastic measures, LCSS is most suitable for our requirements, because it allows for partial comparison and shifting on the time axis. We combine k-medoids with LCSS and use it as a competing approach in the experiments.

Another type of time series clustering methods, designed for applications in which single, long time series are mined for frequently appearing subsequences, are subsequence clustering and motif discovery [LKLP02]. Since we are interested in patterns that occur in several time series at similar points in time and not in patterns that occur in a single time series at arbitrary positions, they cannot be applied in our scenario.

**Subspace Clustering and Triclustering.** Subspace clustering was introduced for high-dimensional (non-temporal) vector data (cf. Chapter 2). Since subspace clustering is simultaneous clustering of the objects and dimensions of a dataset, it can also be denoted as 2D clustering. When subspace clustering is applied to 3D data (objects, dimensions, points in time), the time series of the individual dimensions are concatenated to obtain a 2D space (objects, concatenated time series). The concatenated time series are thus interpreted as high dimensional vector data. While subspace clustering is good for handling noisy data and excluding irrelevant points in time, there...
3.2. Comparison with Related Work

Figure 3.5: Relevance matrix showing a single cluster that can be obtained by a Triclustering algorithm. The relevant points in time in each dimension are marked in black, i.e. they are relevant for all time series of the cluster.

are problems when it is applied to temporal data: subspace clustering cannot exploit the natural correlation between subsequent points in time, i.e. temporal coherence is lost.

Accordingly, for temporal data, Triclustering was introduced [JPR+04, ZZ05, SAG10], simultaneously clustering objects, dimensions, and points in time. Another type of Triclustering approaches are for clustering two related datasets together [AB08, HB10], which is a different problem than the one tackled in this chapter. In general, Triclustering approaches are very rigid with respect to the types of clusters they can find: A cluster is defined by a set of objects, dimensions, and points in time [ZZ05] or intervals [JPR+04, SAG10]; therefore, the points in time or intervals hold for all objects and dimensions of a cluster. This is illustrated in the relevance matrix in Figure 3.5, which abstracts from individual time series and shows the points in time and dimensions that are relevant for the time series of a single cluster. In contrast, our approach can mine clusters where each dimension has different, independent relevant intervals.

The 2D and 3D clustering approaches listed above are not well suited for real world time series data, as it was explained in the introduction: All of them are directly based on the principle of subspace clustering, and thus cannot compare different points in time, i.e. there is no possibility of compensating misalignments between time series, and they have substantial problems when coping with deviating values that only occur in individual time series. In this work, we introduce effective concepts for misalignment compensation and handling of deviating values.
3.3 Robust, Effective Clustering of Multivariate Time Series

We introduce our approach in four steps. We formalize the basic definition of an interval-based cluster model in Section 3.3.1, allowing for clusters that are neither bound to the full dimensional nor temporal extent of the data domain. Based on this, we present an extension in Section 3.3.2 that enables individual shifts of time series, maximizing the similarity between the time series contained in a single cluster. In Section 3.3.3 we increase the robustness by introducing a method for noise handling. In Section 3.3.4 we discuss how a clustering, i.e. a cluster set, is obtained. A dataset can contain an abundance of possible clusters satisfying the conditions of the final cluster model. We are interested in a partitioning clustering that gives a precise and concise description of the dataset: we formalize a clustering that realizes a trade-off between the number and the compactness of clusters.

**Basic definitions.** As input we assume a database \( DB \) of multivariate time series. In this chapter, a time series \( o \in DB \) is defined as \( o = (o[1], \ldots, o[T_{\text{max}}]) \), with \( o[t] \in \mathbb{R}^{D_{\text{max}}} \). \( o[t, d] \in \mathbb{R} \) refers to the value at point in time \( t \in T \) and in dimension \( d \in D \). \( D = \{1, \ldots, D_{\text{max}}\} \) denotes the set of all dimensions, and \( T = \{1, \ldots, T_{\text{max}}\} \) denotes the set of all points in time. Accordingly, we assume a discrete segmentation of time with uniform sampling intervals for the time series.

There are datasets that contain time series of different lengths. Generally, our method can handle such data by design, as we will later explain. In that case, \( T_{\text{max}} \) is the length of the longest time series. Alternatively, one could scale all time series to the same length.

### 3.3.1 Interval-Based Subspace Cluster

In multivariate time series data, patterns can exist in dimensional or temporal subsets of the data domain. Triclustering approaches [JPRt04, SAG10, ZZ05] cope with such data by defining clusters by tuples \((O, S, T)\): The objects \( O \) show a similar behavior in subspace \( S \subseteq D \) and temporal subset \( T \subseteq T \). A drawback is that each selected dimension \( d \in S \) has to be relevant for each
point in time \( t \in T \). In this section, we provide a more flexible concept, laying the foundation for the following sections. According to the new concept, per dimension an individual set of intervals is selected, in which the time series are similar in. Such (relevant) intervals are termed cluster intervals.

**Definition 3.1** A *Cluster Interval* is bound to a specific dimension and is defined by start and end position in that dimension. That is, a cluster interval \( I \) is a tuple \( I := (I_d, I_{\text{start}}, I_{\text{end}}) \) with \( I_d \in \mathcal{D}, I_{\text{start}} \in T, I_{\text{end}} \in T, \) and \( I_{\text{start}} < I_{\text{end}} \). Accordingly,

- the set of all points in time contained in an interval \( I \) is: \( I_{\text{pt}} := \{ t \in T \mid I_{\text{start}} \leq t \leq I_{\text{end}} \} \). In the following, the tuple \((I, t)\) consisting of an interval \( I \) and a point in time \( t \) with \( t \in I_{\text{pt}} \) is denoted as an Interval Point.
- the length of an interval \( I \) is defined by:
  \[ I_{\text{length}} := I_{\text{end}} - I_{\text{start}} + 1 = \left| I_{\text{pt}} \right| \]

Since \( I_{\text{start}} < I_{\text{end}} \), it holds that \( I_{\text{length}} > 1 \); that is, we only consider intervals that actually represent a temporal extent and not just a single isolated point in time, in which the values could be similar by chance.

A time series subspace cluster is a set of time series combined with a set of cluster intervals, in which these time series are similar. The combination of the individual intervals has to ensure the cluster’s temporal coherence.

**Definition 3.2** A *Time Series Subspace Cluster* \( C := (C_O, C_I) \) is defined by a time series set \( C_O \subseteq \mathcal{D}B \) with \( |C_O| > 1 \), interval set \( C_I := \{I_1, \ldots, I_m\} \), and conditions:

- The intervals per dimension are overlap-free, i.e.
  \( \forall I^i, I^j \in C_I \) with \( i \neq j \) and \( I^i_d = I^j_d : \ I^i_{\text{pt}} \cap I^j_{\text{pt}} = \emptyset \).
- The cluster is Temporal Coherent, i.e. the combined set of points in time over all dimensions \( C_{\text{ext}} := \bigcup_{i=1}^{m} I^i_{\text{pt}} \) is a single connected interval:
  \( \exists t \in T \) such that \( \{t, \ldots, t + |C_{\text{ext}}| - 1\} = C_{\text{ext}} \).
- All interval points fulfill a Cluster Property:
  \( \forall I \in C_I : \forall t \in I_{\text{pt}} : \forall o_i, o_j \in C_O : (o_i, o_j) \in \text{compact}_t^{I_{\text{pt}}} \)

i.e., the cluster has to be compact in all interval points. The compactness is defined in the following Definition 3.3.
Overlap-free intervals are required for the clustering process, in which time series are compared with respect to the cluster intervals of a specific cluster (cf. Sec. 3.3.4, Def. 3.9). If tuples would be used several times, their influence would not be equal and some could dominate the similarity comparison.

The cluster property defines how the similarity between time series in the cluster is measured and how similar they need to be for inclusion in the same cluster (for a specific cluster interval). We use compactness as cluster property because it has shown to be effective in other subspace clustering methods [PJAM02].

**Definition 3.3** The Compactness Relation for a point in time \( t \) and dimension \( d \) is based on the maximum norm:

\[
(o_i, o_j) \in \text{compact}^d_t \iff \left| o_i[t, d] - o_j[t, d] \right| \leq w
\]

with compactness parameter \( w \). We analyze the influence of the compactness parameter in the experiments.

In the experiments, we will compare this baseline method introduced in this section† with the advanced version described in the following sections. The crucial part in our cluster definition is the cluster property. In the following steps, we replace this property to obtain a more effective and robust time series subspace clustering technique.

### 3.3.2 Effective Compensation for Misalignments

A problem of real world time series datasets is the possible misalignment between time series pairs; often these time series are very similar but when clustered, the misalignments cause the time series to be placed in different clusters. An effective clustering algorithm has to compensate for these misalignments in the decision whether two time series are clustered together. Fullspace algorithms like k-medoids can compensate by using an elastic distance function like DTW or LCSS. Our approach, however, is based on sub-

---

†The baseline method has been published in the Proceedings of the 16th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2012) [KGHS12b].
space clustering. Until now, no subspace approach exists that can compare different dimensions in the clustering process. This is perfectly reasonable for non-temporal vector data, but a drawback for time series clustering: in the worst case, a time series is lost for a cluster because the misalignment obscures the similarity to the other time series contained in the cluster; if the time series would be included in the cluster it would no longer be compact.

Our novel approach combines the idea of subspace clustering with the idea of maximizing the similarity of time series in each cluster by individual shifts on the time axis. More concretely, in the clustering process, we shift the time series of a cluster on the time axis such that the compactness in the cluster’s current cluster intervals is maximized. The result of our approach for the Spoken Arabic Digit is shown in Figure 3.6, where the \(9^{th}\) dimension of one of the found clusters is plotted. The left side shows the alignment on which the clustering is based: misalignments are compensated by individual shifts. The right side shows the same set of time series, but without the compensation. In this scenario, the cluster is much less compact in the cluster intervals. Accordingly, without the compensation, it is likely that not all of these time series would have been clustered together.

In the following, we formalize our method. Each time series of a cluster is shifted by an individual number of points in time, achieving compactness in that cluster.
Definition 3.4 The Shift of a time series \( o \) is denoted \( \Delta_o \in \Delta \), where \( \Delta \) is the set of all possible shifts \( \Delta := \{ s \in \mathbb{N} \mid -\delta_{\text{max}} \leq s \leq +\delta_{\text{max}} \} \) for a maximal shift value of \( \delta_{\text{max}} \).

The maximal shift parameter is a similar concept as the offset parameter of LCSS (cf. Section 3.2) or the bandwidth constraint of Dynamic Time Warping (cf. Chapter 8). For DTW, it was shown [RK05] that too much bandwidth can result in decreased quality of data mining results. We made similar observations, and we evaluate the influence of \( \delta_{\text{max}} \) in the experiments.

The aforementioned compactness relation (cf. Definition 3.3) is no longer applicable. We introduce a new compactness relation in which time series are shifted before the compactness is computed.

Definition 3.5 The Shift-Based Compactness Relation for a point in time \( t \) and a dimension \( d \) is defined by

\[
(o_i, o_j, \Delta_i, \Delta_j) \in \text{compact}_t^d \iff \left| o_i[t + \Delta_i, d] - o_j[t + \Delta_j, d] \right| \leq w
\]

with compactness parameter \( w \).

A set of time series combined with a set of cluster intervals is a cluster if it is compact in all its interval points. In our new, shift-based approach, this is fulfilled if there exists at least one set of time series shifts such that the cluster is compact in all its interval points.

Definition 3.6 A cluster \( C := (C_O, C_I) \) fulfills the Shift-Based Cluster Property iff

\[
(\forall (\Delta_1, \ldots, \Delta_{|C_O|}) \in \Delta_{|C_O|}: \forall I \in C_I: \forall t \in I_{pt}: \forall o_i, o_j \in C_O:

(o_i, o_j, \Delta_i, \Delta_j) \in \text{compact}_t^d \lor (t + \Delta_i) \notin T \lor (t + \Delta_j) \notin T
\]

i.e., there needs to be a set of shifts such that the cluster is compact in the cluster’s interval points. In specific cases (e.g., \( (t + \Delta_i) \notin T \)) we do not require compactness, as explained below.

When a time series is shifted or time series of different lengths are clustered, there are points in time of the temporal domain \( T \) at which no value exists for this time series; we declare these points in time as invalid. For example,
the time series $b$ in Figure 3.7 has no values for the last points in time of the cluster interval on the right.

These cases only occur at the bounds of the temporal domain. If caused by shifting of time series, then they are even limited by the maximal allowed shift. It is unlikely that all time series have the same invalid positions; accordingly, these areas can contain relevant patterns in the remaining (valid) time series. A complete exclusion of cluster intervals covering these areas could therefore result losing cluster intervals that discriminate the cluster from others. Our cluster model explicitly allows such cluster intervals and excludes only those object pairs from the compactness verification that have invalid positions.

**Relationship to elastic similarity measures.** Fullspace clustering approaches can be combined with elastic distance measures like LCSS (cf. Section 3.2). When two time series are compared by an elastic measure, this time series *pair* has an individual alignment of the points in time. This is in contrast to our approach, where we have one individual shift per *single* time series (Definition 3.6: $\exists(\Delta_1, \ldots, \Delta_{|CO|}) \ldots \forall o_i, o_j \ldots$). Accordingly, one could expect that using individual shifts for each pair of time series would be more effective (i.e., $\forall o_i, o_j \ldots \exists(\Delta_{o_i}, \Delta_{o_j}) \ldots$). This is true for fullspace clustering, but not for temporal subspace clustering. By allowing individual shifts per time series pair, these two time series are shifted only with respect to each other (and not with respect to all time series from the cluster) such that an arbitrary pattern from these two time series is used to achieve compactness in the cluster intervals. This is repeated for every pair, and thus in each pair a
different pattern could be used. Accordingly, the cluster no longer represents a specific pattern, which is a contradiction to the idea of clustering.

We implemented variants of our approach using measures like LCSS to compensate misalignments. However, results based on these more complex (and less efficient) distance measures did not show the same effectiveness as the shift-based method.

The presented model compensates for misalignments between similar time series in a cluster. In the following, we present an improvement that enables robustness with respect to deviating values in single time instances.

3.3.3 Robust Handling of Deviating Values

Deviating values, as for example caused by measurement errors, are unavoidable in real datasets. Values are called deviating, if they only occur in a few time series of a cluster and strongly deviate from the distribution of the other time series' values. Subspace clustering algorithms exclude dimensions (or intervals in our interval-based cluster paradigm) for whole clusters, i.e. for all contained time series. The dimensions are either relevant or non-relevant. There is no concept of relevant dimensions that contain some deviating values. Since our model so far is based on subspace clustering, it either has to remove a time series with a deviating value from the cluster, or the corresponding position in the cluster interval has to be considered non-relevant, splitting up the interval into two subintervals. In the worst case, the cluster loses a pattern that discriminates it from other clusters.

Technically, this is attributable to the cluster property in Definition 3.6, which considers interval points as non-compact in case of deviating values. In the following, we extend Definition 3.6 such that a small number of deviating values does no longer force a relevant point in time to be categorized as non-relevant. Concretely, the improved model allows for a certain fraction of deviating values per point in time in the cluster intervals.

In the new cluster property, we assume that the object values in a cluster's interval point \((I,t)\) follow a normal distribution with a variance based on the compactness parameter \(w\). Concretely, we have a normal distribution
Figure 3.8: Illustration of the assumed normal distribution.

$\mathcal{N}(\mu_{it}, \sigma^2)$, where $\mu_{it}$ is the center of cluster interval $I$ at point in time $t$, and the standard deviation $\sigma$ corresponds to $w/T$. Figure 3.8 illustrates this. We define an interval point to be compact, if the probability of its compactness is larger than the probability that two arbitrary points $x, y \in \mathcal{N}(\mu_{it}, \sigma^2)$ are compact, i.e. $|x - y| \leq w$.

**Definition 3.7** A cluster $C := (C_O, C_I)$ fulfills the **Shift and Deviation Robust Cluster Property** iff $\exists (\Delta_1, \ldots, \Delta_{|C_O|}) \in \Delta^{|C_O|} : \forall I \in C_I : \forall t \in I_{pt} :$

$$P \left( (o_i, o_j, \Delta_i, \Delta_j) \in \text{compact}_{it} := (a_i, a_j) \in \text{valid}, \ o_i \neq o_j \right) \geq P \left( |x - y| \leq w \mid x, y \in \mathcal{N}(\mu_{it}, (w/2)^2), \ x \neq y \right)$$

where valid := \{o_i \in C_O \mid t + \Delta_i \in T\} is the set of objects valid in the interval point $(I, t)$.

The probability of an interval point $(I, t)$ to be compact is the ratio of compact valid object pairs in $(I, t)$ to the number of all valid object pairs $\binom{|\text{valid}|}{2}$ in $(I, t)$. The probability that $x, y \in \mathcal{N}(\mu_{it}, \sigma^2)$ fulfill $|x - y| \leq w$ is independent of the actual $\mu_{it}$. Interestingly, in case of $\sigma = w/2$ it is also independent of $w$. When analytically solved, we obtain a probability of about 85%.

Accordingly, Definition 3.7 holds if there is a set of shifts such that in all interval points compactness is achieved, with an allowed violation in 15% of the time series pairs.

This completes our final cluster model. In the following, we introduce the definition of a corresponding cluster set.
3.3.4 Clustering Model

In this section we formalize a set of clusters representing the patterns in the analyzed dataset $DB$. We are interested in a partitioning that gives a precise and concise description of the dataset. Accordingly, we have a trade-off between generalization (number of clusters) and precision (compactness of single clusters). We realize this trade-off with the clustering costs. The costs are the sum of the average cluster diameters ($\varnothing_{\text{Cluster}}$ w.r.t. Definition 3.10 below), reflecting both the number of clusters and the cluster compactness.

**Definition 3.8** The Costs of a Clustering $M := \{C^1, \ldots, C^n\}$ are

$$\text{cost}(M) := \sum_{C \in M} \varnothing_{\text{Cluster}}(C)$$

The lower the costs are, the better is the trade-off between the number of clusters and the cluster diameters. The costs are not monotonic in the number of clusters: by transferring time series from clusters $\{C^1, \ldots, C^n\}$ to an additional cluster $C^{n+1}$, the costs of clustering $M' := \{C^1, \ldots, C^n, C^{n+1}\}$ with object sets $\{C^1 \setminus C^{n+1}, \ldots, C^n \setminus C^{n+1}, C^{n+1}\}$ can be larger (the additional cluster outweighs smaller cluster diameters), smaller (the additional cluster is counterbalanced by smaller cluster diameters), or even the same as for $M$.

The average diameter of a cluster reflects its compactness and is based on the average pairwise distance of the contained time series. This distance is constrained to the cluster intervals and uses the shifted time series.

**Definition 3.9** The Normalized Shift Distance between time series $o_i, o_j$ for a cluster $C$ and its valid interval points

$$\text{intp} := \{(I, t) \mid I \in C, t \in I, (t + \Delta_i) \in T, (t + \Delta_j) \in T\}$$

is defined as

$$d_C(o_i, o_j, \Delta_i, \Delta_j) := \frac{1}{|\text{intp}|} \sum_{(I, t) \in \text{intp}} |o_i[t + \Delta_i, I_d] - o_j[t + \Delta_j, I_d]|$$

Based on this distance, we define the cluster diameter which is used for the clustering costs in Definition 3.8.
Definition 3.10 The **Cluster Diameter** is the minimal average pairwise time series shift distance in that cluster:

$$\varnothing_{\text{Cluster}}(C) := \min_{(\Delta_1, \ldots, \Delta_{|C_o|}) \in \Delta_C} \frac{1}{\binom{|C_o|}{2}} \sum_{o_i, o_j \in C_o, o_i \neq o_j} d_C(o_i, o_j, \Delta_i, \Delta_j)$$

with $\Delta_C \subseteq \Delta_{|C_o|}$, all shift sets that yield a fulfilled cluster property according to Definition 3.7. Since $C$ is a valid cluster, it holds that $|\Delta_C| \geq 1$. The minimal cluster size in Definition 3.2 ($|C_o| > 1$) ensures the normalization factor is not 0, i.e. $\left(\frac{|C_o|}{2}\right) \geq 1$.

We know by Definitions 3.6 and 3.7 that there is at least one shift set such that $C$ is compact in its cluster intervals. In the final clustering result, however, we are searching for a set of maximally compact clusters. Accordingly, for determining the cluster diameter, we minimize over all shifts sets that yield a compact cluster, reducing the clustering costs in Definition 3.8.

With the cluster diameter, the definition of the clustering costs is complete. The optimal clustering is defined by demanding three conditions.

Definition 3.11 The **Cost-Minimizing Clustering** with respect to our cluster model is a set of clusters $M := \{C_1, \ldots, C_n\}$ with

1. each $C \in M$ is a valid cluster according to Definition 3.2.
2. complete, disjoint object coverage: $\bigcup_{C \in M} C_o = DB$.
3. minimal clustering costs: For all clusterings $M'$ fulfilling conditions (1.) & (2.) it holds that $\text{costs}(M) \leq \text{costs}(M')$.

We chose to realize a partitioning clustering approach, since the real world time series data we analyzed were of this kind.

With this definition of a clustering, the formalization of our novel clustering model for multivariate time series is complete. The determination of the clustering in Definition 3.11 is an optimization problem. As for other partitioning clustering approaches, it can be proven that the problem is NP-hard [ADHP09].
3.4 Efficient Computation

We present an efficient algorithm that determines an approximate solution with respect to our proposed clustering model.

Algorithm 3.1 shows the corresponding pseudocode. In the algorithm, each cluster $C$ is represented by a representative $C_p \in C_O$. The cluster intervals are specified with respect to $C_p$; that is, without loss of generality, we assume that $\Delta C_p = 0$. The remaining time series $C_O \setminus \{C_p\}$ are possibly shifted. The current set of shifts for a cluster $C$ is denoted as $\Delta C$.

Our algorithm is divided into three phases. The first phase is the cluster generation phase, in which a superset of the final clustering is generated. The following refinement phase eliminates redundant clusters from this set. In the last phase, the final assignment of the time series to the clusters takes place. In the next three paragraphs these phases are explained. The representatives, the cluster intervals, and the shifts are constantly updated during the three phases. The last two paragraphs of this section explain how these recomputations are achieved.

**Phase 1: Cluster generation.** In this first phase (l. 2-19), the objective is to find a set of clusters, or more concretely cluster representatives, corresponding to a superset of the actual clustering result. The cluster representatives are generated iteratively, and during the iterations the already obtained representatives are simultaneously refined. The representatives are maximally different (l. 4), such that different patterns in the data are captured. At the beginning, the representatives are rough approximations of the final clusters. While generating more clusters, already generated clusters are simultaneously enlarged by adding more time series (l. 10-17). The added time series are drawn from the whole database $DB$ (independently for each cluster). The adding order (l. 11) is based on the normalized shift distance (Def. 3.9) to the cluster representative, i.e. $\min_{\Delta \in \Delta} d_C(o, C_p, \Delta, \Delta C_p = 0)$. By using this distance, we ensure a minimal enlargement of the cluster diameter (Def. 3.10) and thus low clustering costs (Def. 3.8 & 3.11). By drawing the time series independently for each cluster, we ensure time series assigned to a wrong cluster can still be added to the correct one.
3.4. Efficient Computation

Algorithm 3.1: Algorithm for approximate solutions.

1. $M \leftarrow \emptyset$; $DB_{cand} \leftarrow DB$; $batchSize = 3$

//Iterative cluster generation:
2. repeat
3. //new cluster $C$ with $C_p, C_O$
4. if ($M \neq \emptyset$) then
5. $C_p \leftarrow \max_{o \in DB_{cand}} \{\sum_{C_j \in M} L_1(o, C_j^{p})\}$
6. else
7. $C_p \leftarrow \text{random}(DB_{cand})$
8. $M.add(C)$; $DB_{cand}.remove(C_p)$
9. $C_O \leftarrow \{C_p\}$
10. $C.adaptIntervals()$ //here: fullspace
11. foreach $C_j \in M$ do //grow clusters
12. $C_j.orderTimeSeriesCandidates()$
13. $C_j.addTimeSeriesCandidates_{batchSize}()$
14. while $\exists C_i, C_j \in M, i \neq j$ and $|C_i^{O} \cap C_j^{O}| > 1$ do
15. discardSmallerCluster($C_i, C_j$)
16. $C_j.recomputeShifts()$
17. $C_j.adaptIntervals(85\%)$
18. $batchSize ++$
19. until ($|M|$ converges $\lor DB_{cand} = \emptyset$)

//Refinement phase:
20. foreach $C \in M$ do
21. $C.FillUpClusters(maxSize)$
22. while $\exists C_i, C_j, i \neq j$ with $|C_i^{O} \cap C_j^{O}| > |C_i^{O} \setminus C_j^{O}|$ do
23. randomlyDiscardCluster($C_i, C_j$)
24. foreach $C \in M$ do
25. $C.recomputeShifts()$
26. $C.adaptIntervals()$

//Final Object Assignment:
27. return generatePartitioningClustering($M$)
We want to find overlap-free clusters (Def. 3.11). Accordingly, we need to avoid clusters whose cluster representatives are based on overlapping object sets. In case two clusters overlap, we discard the smaller cluster (l. 14).

Time series are added in batches (l. 12), and the batch size is constantly increased with each iteration of this phase (l. 18). This reflects that at the beginning of this phase, cluster representatives are unstable and potentially undergo many changes; adding large batches of time series based on a wrong representative would yield a stabilization of clusters possibly not contained in the data. The initial batch size of 3 is a trade-off between a sufficiently small size allowing flexible cluster changes and enough time series for a fast rejection of similar clusters (due to overlap); it was determined by preliminary experiments.

By adding time series to a cluster, the cluster representative, shifts, and cluster intervals are refined. Since recomputations of representatives and shifts are costly, more established clusters are recomputed in larger intervals. This is done by an exponential function (l. 15), ensuring a recomputation each time the cluster size \( |C_O| \) reaches a power of 2 (2, 4, \ldots) after adding a batch. Accordingly, larger and thus more stable cluster are less often refined and a substantial runtime improvement is achieved. Since it is a less expensive operation, the intervals of a cluster are adapted after each batch insert (l. 17).

The objective of phase 1 is to generate a superset of the final clusters. Termination of our algorithm is guaranteed, since the cluster generation phase is stopped when all cluster candidates have been probed or when the number of clusters in \( M \) converges (l. 19). The latter is reasonable, since with a sufficient number of iterations, the number of cluster candidates showing a difference to the already existing cluster representatives decreases substantially. Accordingly, overlap between established clusters and newly generated ones is highly probable, and new ones will be instantly discarded. Technically, this convergence is assumed when the cumulated increase in clusters over the last 10 iterations reaches 0. Figure 3.9 shows the number of clusters and the cumulated increase for the Spoken Arabic Digit dataset. In this example, the cluster generation phase terminates after 38 iterations. The
3.4. Efficient Computation

Figure 3.9: Convergence of the main phase on Spoken Arabic Digit dataset. Our algorithm stops after 38 iterations.

The figure shows that if more iterations would be performed, there would be no increase in the cluster number.

**Phase 2: Refinement.** In many datasets, the obtained representatives are a good reflection of the final clusters. In some datasets, however, multiple clusters are generated that only represent parts of a single cluster. This is also the case in Figure 3.9: the dataset has 10 clusters, but 20 are generated. The problem in these datasets is that while there is high inter-cluster dissimilarity, there is also some intra-cluster dissimilarity. Dependent on the initial cluster representatives, two or more clusters are generated for single clusters. The objective of the refinement phase is to retain only one cluster instead of the multiple clusters. To detect these cases, we use a similar technique as in the generation phase. Independently for each cluster (obtained in the generation phase and reset to \( C_O = \{C_p\} \), with the most recent representative \( C_p \)), we add time series from \( DB \) according to the normalized shift distance. If sufficiently many time series are added to the multiple clusters that should be a single cluster, the probability increases that the same time series are added to these clusters; that is, the difference between the corresponding representatives makes no longer a difference. To obtain an equal weighting, the number of time series added to each cluster is the same (\( maxSize \)), and is the maximal obtained cluster size from the cluster generation phase. Afterwards, for each pair of clusters sharing more than half of the objects, one is discarded. In the final assignment phase, the remaining cluster will absorb the time series from the discarded cluster. Phase 2 is completed by a recomputation of the representatives, shifts, and intervals.

**Phase 3: Final assignment.** Our approach generates a partitioning clustering. In this last step, the cluster object sets \( C_O \) are reset. Then, based on
the normalized shift distance, each $o \in DB$ is assigned to the best-fitting cluster representative. Accordingly, a disjoint and complete assignment of time series to clusters is realized.

**Shift and Representative Recomputation.** (l. 16,25) At cluster initialization and during the growing phase, it is unlikely that the best representative is chosen. The best cluster representative is a time series to which the other time series in the cluster need a minimal shift and that is close to the cluster's center. A representative not positioned near the center can absorb time series belonging to other clusters in the growing phase. A representative not minimizing the shifts can force time series to be shifted near the maximal shift, decreasing the potential to absorb time series in the growing phase that retain compactness in the cluster intervals. Accordingly, based on the distribution of shifts to the current representative, we chose a new representative that potentially minimizes the shift to this new representative. Concretely, we obtain the set $\{ o \in C_0 \mid \Delta_o = \left\lceil \frac{1}{|C_0 \setminus \{C_p\}|} \sum_{o \in \Delta_c} \Delta_o \right\rceil \}$, containing all time series whose shifts are similar to the average shift of the cluster. From this set, we chose the time series that is most similar ($L_1$ distance) to the cluster's center as a new representative, avoiding a representative that could absorb time series from other clusters. After obtaining the new representative, the shifts of the cluster are recomputed with respect to the new representative. In the algorithm, this recomputation is followed by interval recomputation and time series candidate reordering (based on new shifts and intervals).

**Interval Recomputation.** (l. 9,17,26) When additional time series are added to a cluster or the cluster's representative and the corresponding shifts were adapted, the cluster intervals need to be recomputed to better reflect the cluster. We obtain the new cluster intervals by verifying for each tuple of dimension $d$ and point in time $t$ whether 85% of the valid object pairs at $t$ in $d$ are compact (cf. Def. 3.7), enabling the robust handling of deviating values. The compact tuples form the adapted cluster intervals. Cluster intervals that prevent temporal coherence (cf. Def. 3.2) are discarded (in case of several possibilities, the larger intervals are kept). When a new cluster is generated, it only consists of its representative. In this case, the cluster intervals span all dimensions and all points in time (l. 9).
3.5 Experiments

We evaluate our method, which is denoted as Robust Time Series Subspace Clustering (RTSC), in comparison to five competing approaches. From the area of fullspace clustering we chose a k-means that uses statistical features to cope with multivariate time series [WWW07], and k-medoids combined with LCSS [VKG02] as distance measure to allow for partial comparison. From subspace clustering, we included Proclus [APW+99] and MineClus [YM03]. From Triclustering, we included MIC [SAG10]. We also compared to TriCluster [ZZ05] (from the author’s webpage), but it either delivered no result or an accuracy below 3%. The used synthetic data was too large for MineClus, but we could include it in nearly all real data experiments; this issue is discussed in the paragraph on efficiency. The implementation of MIC, which was provided by the authors, did crash in some experiments. To allow a better analysis of our approach, we also included a version where we did not use the compensation for misalignments and deviating values. The cluster model and algorithm for this basic version can find in our corresponding publication [KGHS12b], and in the experiments, this version is denoted as Temporal Subspace Clustering (TimeSC). We evaluated our approach on five real world datasets, described in the corresponding paragraph. If not stated otherwise, we use the following settings for the synthetic data generator: The dataspace has an extent of \([-100, +100]\), \(T_{\text{max}}=200\), \(D_{\text{max}}=10\), cluster length is 100, the number of relevant dimensions per cluster is 5, the number of clusters is 10, the number of time series per cluster is 50. The first time series of a cluster is generated by random walk in the cluster intervals. The remaining time series are then added with a variance of 30. Afterwards, shift and deviating values are added randomly. The maximal shift is 5, and there are 600 deviating values per cluster (distributed over the interval points). In our approach, we use \(w=30\) for the compactness parameter, corresponding to 15% of the dataspace’s extent. The maximal shift is set to \(\delta_{\text{max}}=15\); this choice is discussed in the next paragraph. The parameters of the approaches were tuned to obtain highest quality. The experiments were performed on AMD Opteron servers with 2.2GHz and 256GB RAM. The clustering qual-
ity in the experiments on synthetic data was measured with the CE and the E4SC measure, both being extensions of the F1 measure such that subspaces are considered [GFM+11]. Since they showed comparable results, we only included the E4SC measure. The ground truth of the real world data does not specify the data’s subspaces; we use the F1 measure to determine the clustering quality in these experiments. The usage of the F1 measure for clustering is explained in [GFM+11]. All values are averages of three runs.

![Graph](image_url)

**Figure 3.10: Evaluation of cluster model parameters.**

**Influence of model parameters.** We start by analyzing the influence of the two parameters of our clustering model. Figure 3.10(a) shows the robustness of our approach with respect to the compactness parameter \( w \). As previously mentioned, the synthetic data generator uses \( w = 30 \). Obviously, the choice of \( w \) is important. However, the figure shows that there is a wide range in which the clustering qualities are stable (in this case: \( 20 \leq w \leq 40 \)), making the parameter determination easier for the user. Figure 3.10(b) shows results for a fixed dataset (time series are shifted by maximal 10 points in time) for a varying maximal shift \( \delta_{\text{max}} \). As expected, the clustering quality increases if we allow more shift in the clustering model, until the maximal quality is obtained at \( \delta_{\text{max}} = 10 \). Afterwards, the clustering quality is stable for a wide range until it starts to decrease. This illustrates that the choice of \( \delta_{\text{max}} \) is not critical as long as it is not chosen too small. However, the figure shows on the secondary axis that large \( \delta_{\text{max}} \) also incur high runtimes. We chose to use \( \delta_{\text{max}} = 15 \) as standard.
3.5. Experiments

![Figure 3.11: Effectiveness on synthetic data.](image)

**Effectiveness on synthetic data.** We compare our method to the competing solutions with respect to several aspects. Figure 3.11(a) shows how the approaches cope with a varying time series length. The plot shows that all approaches show stable clustering qualities. Our new approach outperforms the competing solutions by more than 30%. From the other subspace clustering results, Proclus shows the best results, followed by TimeSC and MIC. The low performance of MIC is surprising, as MIC is designed for temporal data. The large gap from our approach to Proclus can be explained by the temporal nature of the data which Proclus cannot handle. The low quality obtained by TimeSC is caused by misalignments and deviating values, as we will show later. The fullspace approaches deliver clusterings of low quality. Surprisingly, the LCSS-based k-medoids cannot counterbalance the cluster intervals and misalignments in the data. An experiment in which we analyzed an increasing number of dimensions showed very similar ratios.

In Figure 3.11(b), we analyze the influence of varying relevant dimensions per cluster. The results of our method and the other subspace clustering approaches are very similar to the experiment in Figure 3.11(a). As expected, the clustering qualities of the fullspace approaches improve with an increasing number of relevant dimensions. An experiment with a varying number of relevant points in time showed similar results.

The influence of shift in the data is analyzed in Figure 3.12(a). The plot shows that with increasing shift, the clustering quality of TimeSC constantly decreases, while RTSC can compensate these misalignments until a shift of
15 is reached. Afterwards, the clustering quality yielded by our approach also decreases. The influence of an increasing number of deviating values per cluster is analyzed in Figure 3.12(b). As for misalignments in Figure 3.12(a), our approach is robust to deviating values until their number grows very large. Both experiments indicate why in the previous experiments TimeSC performs much worse than Proclus: it is very prone to misalignments and deviating values, which occurred in the synthetic data in Figure 3.11. The other approaches obtain low quality clusterings as in the previous experiments, caused by the data’s temporal subspace aspects.

**Efficiency on synthetic data.** In the experiments in Figure 3.13, we increase the number of clusters and dimensions, and we included MineClus (discussed on page 41). Both experiments show that our RTSC is much faster than Proclus, MIC, and MineClus. The values for MineClus are based on downscaled ($T_{\text{max}}=50, D_{\text{max}}=6$, 10 time series per cluster) versions of the datasets; the real runtimes are much higher. The obtained clustering quality of MineClus are below the ones achieved by Proclus. Only TimeSC and the k-means variant are faster than RTSC; their effectiveness, however, is much worse as we demonstrated before.

**Real world data.** We selected three multivariate datasets that seemed suitable for our approach (for example, due to misalignments and/or deviating values; cf. Figures 3.1, 3.2, 3.6) from the UCI Machine Learning Repository [FA10]. The datasets are Spoken Arabic Digit ($D_{\text{max}} = 13$, $T_{\text{max}} = 93$, $|DB|= 8,800$, 10 clusters), Japanese vowels ($D_{\text{max}} = 12$, $T_{\text{max}} = 25$, $|DB|=640$, $|DB|=640$, $|DB|=640$, $|DB|=640$, $|DB|=640$).
3.5. Experiments

Figure 3.13: Efficiency on synthetic data (log. scale).

9 clusters), and Robot Execution Failures ($D_{\text{max}}=6$, $T_{\text{max}}=15$, $|DB|=164$, 5 clusters). Figures 3.14(a-c) show the results. In some cases, MIC & MineClus did not finish. In all three datasets, our approach outperforms the competing approaches by a substantial margin, showing the effectiveness of our introduced concepts. Our approach is not limited to multivariate data, as Figures 3.14(d-e) illustrate. We evaluate our approach on the univariate Trace [KZH+11] and the Mallet [YK09] datasets. Again, our approach clearly outperforms the competing solutions.

Figure 3.14: Real world data: multivariate (a-c) and univariate (d-e).
3.6 Conclusion

We proposed a novel model for subspace clustering of multivariate time series data. The clusters in our model are formed by individual sets of relevant intervals per dimension, which together fulfill temporal coherence. Misalignments between time series in a cluster are compensated by individual shifting on the time axis. Robustness to deviating values is achieved by allowing a certain fraction of deviating values in each relevant point in time. These novel concepts go beyond existing subspace clustering algorithms, which cannot compare different dimensions and only distinguish between relevant and non-relevant points in time. The concepts enable our method to discover patterns missed by other approaches. Since our model is NP-hard, we proposed an algorithm that yields approximate solutions. It is both effective and efficient, as we showed in the experiments.

This finishes the first part of this thesis, in which we concentrated on the extraction of high-quality clusterings in temporal data. In the next part, we introduce mining techniques for other types of temporal data, namely data streams and spatio-temporal data. In these techniques, we assume several clusterings to be given, and the main focus is on the relationships between these clusterings. In the data stream scenario, we develop an effective method for the evaluation of generated stream clusterings. In spatio-temporal data, we trace evolving subspace clusters over time, with a focus on their evolutions. The effectiveness of both tasks is highly dependent on the applied mapping components that map clusters in one cluster set to the most-fitting counterparts in a second cluster set.
Part II

Effective Cluster Mapping for Temporal Data
Chapter 4

An Evaluation Measure for Clustering on Data Streams

* Due to the ever growing presence of data streams, there has been a considerable amount of research on stream mining algorithms. While many algorithms have been introduced that tackle the problem of clustering on evolving data streams, hardly any attention has been paid to appropriate evaluation measures. Measures developed for static scenarios, namely structural measures and ground-truth-based measures, cannot correctly reflect errors attributable to emerging, splitting, or moving clusters. These situations are inherent to the streaming context due to the dynamic changes in the data distribution.

In this chapter we develop a novel evaluation measure for stream clustering called Cluster Mapping Measure (CMM). CMM effectively indicates different types of errors by taking the important properties of evolving data streams into account. We show in extensive experiments on real and synthetic data that CMM is a robust measure for stream clustering evaluation.

*This chapter has been published in the Proceedings of the 17th ACM Conference on Knowledge Discovery and Data Mining (SIGKDD 2011) [KKJ’11].
4.1 Motivation

Mining patterns from data streams is constantly gaining importance. Streams consist of data tuples that need to be processed as they arrive, and mining these streams is challenging since the data distribution underlying a stream can evolve significantly over time. Stream mining tasks are, for example, classification [BHP+09], association rule mining [MMBA10], or clustering [AHWY03, CEQZ06, KABS09]. In stream clustering, a clustering is constantly adapted to reflect changes in the observed stream. Besides dealing with evolving distributions, stream clustering algorithms have to meet several technical requirements, including limited time, limited memory, and processing the stream in a single pass. A multitude of stream clustering algorithms has been proposed in the literature that satisfy these requirements.

Of major importance is the quality of the resulting clusterings, which can be measured by evaluation measures, also termed criteria, indices, validation measures, or validation indices. Research on evaluation measures for static datasets, i.e. traditional clustering without the streaming context, looks back at a history of more than thirty years. While clustering itself is commonly accepted as a difficult and subjective task [JMF99], the validation of clustering results is even described as the most difficult and frustrating part of cluster analysis [JD88]. This holds for evaluation no matter whether or not a ground truth is available against which to compare the clustering result. The usage of a ground truth can strictly categorize the proposed measures into internal and external measures [BSH+07, LLX+10, SZ08, WXC09]: Internal measures consider only the structure and properties of the clusters, e.g. their compactness or the distance between them. External measures compare the resulting clustering against a ground truth, e.g. punishing an algorithm for putting objects from different ground truth clusters into a single cluster.

The MOA Framework [MOA] is an open source benchmarking software for data stream mining, containing a set of stream clustering algorithms from the literature as well as an extensive collection of evaluation measures. We give a overview over MOA in Section 4.6. The experience from implementing and using MOA showed two major disadvantages of existing evaluation
measures: First, none can properly handle the peculiarities of evolving data streams such as overlap due to merging or drifting clusters or noisy data streams. As a consequence, the measures cannot effectively reflect the occurring errors. Second, the vast majority of evaluation measures achieve suboptimal results even if the ground truth clustering is tested. The Cluster Mapping Measure (CMM) proposed in this chapter overcomes these shortcomings and enables effective evaluation of clustering results on evolving data streams.

The rest of this chapter is structured as follows: We review existing work in Section 4.2. The Cluster Mapping Measure is introduced Section 4.3. We analyze the CMM with respect to error types on evolving streams, different clustering approaches and properties of the data in Section 4.4 and evaluate it in Section 4.5. Section 4.6 gives a brief overview over stream clustering in MOA, and Section 4.7 concludes this chapter.

### 4.2 Comparison with Related Work

We first review existing work on evaluation measures for clustering on static datasets. After that we discuss related work on stream clustering algorithms and identify the evaluation measures employed in the individual approaches.

As mentioned in Section 4.1, evaluation measures can be categorized into internal and external measures, depending on whether or not they employ a ground truth clustering for comparison. A different categorization additionally identifies so called relative measures, comparing the single partitions within a clustering result. This group, however, is less recognized and solutions are often specialized to a specific domain [RLB02, YHR01]; we focus on internal and external measures. Table 4.1 lists (without claiming completeness) a collection of internal and external measures from the literature with references to formulae and more detailed description. Apart from these, other measures exist that deal with fuzzy clustering [BWS06] or focus on finding the best number of clusters [SRS08, WNB+10]. For readability and space reasons we reduced the amount of measures used in our experiments.
Table 4.1: Internal and external evaluation measures.

to the ones highlighted in gray. Our choice of internal measures is based on a study of thirty measures in [Mil81]; the external measures resulted as best choices in three recent studies [BSH'07, SZ08, WXC09].

Stream clustering algorithms can be categorized from different perspectives, e.g. whether convex or arbitrary shapes are found. Convex stream clustering approaches are based on a \( k \)-center clustering [AHWY03, KABS09, OMM+02]. Approaches for arbitrary shaped clusters use kernels [JZC06], fractal dimensions [BC00], or density based methods [CEQZ06, CT07]. In the evaluation of these methods, the used measures are those for static data sets, most often Entropy (Purity), Precision, or the F-measure. In convex approaches, the most prevalent in the literature, often also the sum of squared distances is used (e.g., [AHWY03, OMM+02]), i.e. the compactness of spherical clusters is measured, which does not testify the quality of a clustering.

Most stream clustering algorithms use an internal representation, called micro clustering, that is constantly updated. Only when required, the output (macro) clusterings are generated by an offline component, e.g. k-means.

In summary, we find that despite the amount of work on developing evaluation measures as well as developing stream clustering algorithms, no attempt has been made to meet the requirements of both tasks.
4.3 Cluster Mapping Measure

4.3.1 Preliminaries

In this section we provide formal definitions used throughout this chapter. We concentrate on spherical clusters, which constitute the most prevalent model. It is, for example, also used in the online components of density based algorithm [CT07].

An object arriving on a data stream is represented by a tuple of attribute values. We define a stream as follows:

**Definition 4.1** A Stream $S = \{o_1, o_2, \ldots\}$ is an infinite sequence of objects $o_i$, where each object is a $d$-dimensional tuple of attribute values $o_i = (o_{i1}, \ldots, o_{id})$ with $o_{ij} \in \mathbb{R}$.

In order to give more influence to recent data, objects are assigned an age-depending weight. We exemplarily define the weight using an exponential decay function as in [AHWY03]. Other weighting functions assign a binary weight (sliding window) using either age or cardinality for thresholding.

**Definition 4.2** Let $t_{now}$ be the current time and $t_o$ the arrival time of object $o$ with $t_o \leq t_{now}$. The exponentially decaying Weight of $o$ is $w(o) = \beta^{-\lambda(t_{now}-t_o)}$.

The parameters $\beta$ and $\lambda$ control the shape of the aging function, e.g. if $\beta = 2$ then $1/\lambda$ is the half life of $o$. Apart from the complete stream $S$ we often only consider a subset of the most recent objects that we call horizon.

**Definition 4.3** The Horizon $\mathcal{H}$ for a stream $S$ and threshold $\xi$ is defined as $\mathcal{H} = \{o \in S | w(o) \geq \xi\}$.

To employ a consistent terminology we also formally define clustering and ground truth.

**Definition 4.4** Clustering. A clustering algorithm takes a set $O = \{o_1, \ldots, o_n\}$ of objects as input and returns a cluster set $\mathcal{C} = \{C_1, \ldots, C_k, C_\emptyset\}$. $o \in C_i$ implies that $o$ lies within the cluster boundary of $C_i$ and $C_\emptyset$ contains all unassigned objects. Objects may fall into several clusters.
A ground truth assigns each object to a specific class.

**Definition 4.5 Ground Truth.** For a given object set \( O = \{o_1, \ldots, o_n\} \) a ground truth \( CL = \{Cl_1, \ldots, Cl_l\} \) is a partitioning of \( O \) with \( \forall i \neq j : Cl_i \cap Cl_j = \emptyset \) and \( \bigcup_{i=1}^l Cl_i = O \). The partitions \( Cl_i \) are called classes and \( Cl(o) \) is the class of \( o \). In the presence of noise there is a noise class \( Cl_{\text{noise}} \) with \( CL^* = CL \cup \{Cl_{\text{noise}}\} \) and \( O^* = O \cup Cl_{\text{noise}} \).

Dependent on the applied clustering model, the classes of the ground truth do not directly correspond to ground truth clusters. As already mentioned, in this paper we apply a spherical cluster model, i.e., a cluster boundary is the smallest ball containing all its objects. Accordingly, the boundary of a ground truth class can also contain objects from the other classes.

**Definition 4.6 Ground Truth Clusters.** We define a ground truth cluster \( Cl^*_i \) for a class \( Cl_i \) as the smallest cluster that contains all objects from \( Cl_i \) within its boundary: \( \forall o \in Cl_i : o \in Cl^*_i \). Additionally, \( Cl^*_i \) contains the points of other classes that fall in its boundary, i.e. \( Cl^*_i \supseteq Cl_i \).

As a consequence, two ground truth clusters resulting from two different ground truth classes can overlap, i.e. they are not necessarily disjoint. This ground truth clustering is the best clustering obtainable by a spherical clustering algorithm.

In stream processing, data mining algorithms have to take the special properties of the streaming scenario into account. Likewise, for a reasonable comparison of the algorithms, evaluation measures should also consider these special circumstances:

1. **Aging / decay.** Dealing with this property is probably the simplest task, because faults caused by a clustering algorithm can be weighted by the influence of the corresponding points (cf. Definition 4.2).

2. **Missed points.** Moving clusters yield errors for missed points. These errors should reflect the seriousness, e.g. how close the point is to its actual cluster.
3. **Misplaced points.** Evolution, merging, and splitting of clusters causes overlapping clusters and thereby easily misplaced points. A measure that punishes these misplaced points equally to misplaced points laying outside of any overlapping region does not account for the special circumstances of evolving streams.

4. **Noise.** Including noise in a found cluster is often inevitable in the model of the clustering algorithm and should be accounted for by an effective measure.

Summarizing these properties, three fault cases can be identified that have to be considered in depth, namely missed points, misplaced points, and noise inclusion. The penalty for such errors of stream clustering algorithms should reflect their seriousness and take the age of the points as well as the clustering model into account. CMM is a normalized sum of the penalties for occurring errors that accounts for all aspects mentioned above. Two important prerequisites for the computation are a notion of how well an object fits into a cluster and a mapping from found clusters to ground truth classes. We introduce these prerequisites and CMM in the following section and analyze its behavior in the fault scenarios in Section 4.4.

### 4.3.2 Connectivity

A central concept used throughout CMM is the connectivity between points and clusters. This connectivity states how well the point is connected to the cluster, i.e. how it fits the distribution of the cluster in comparison to the other points. We define the connectivity as a distance-based concept that ranges from 0 to 1, where 0 indicates no connectivity and 1 indicates a strong connectivity. As a prerequisite we define the average $k$-neighborhood distance for points and clusters. $k$ is a locality parameter, and we show in Section 4.5 that it has only marginal influence on CMM effectiveness.

**Definition 4.7 Average $k$-Neighborhood Distance.** The $k$-neighborhood \( knh \) of a point $p$ in a cluster $C_i$ are the $k$ closest points, i.e. \( knh(p, C_i) \subseteq C_i \setminus p \) with $|knh(p, C_i)| \leq k$, and for objects $o_{in} \in knh(p, C_i)$ and $o_{out} \in C_i \setminus knh(p, C_i)$
Figure 4.1: Point connectivity in clusters. $p_1$ and $p_3$ have stronger connections to their clusters than $p_2$.

It holds $\forall o_{in} \forall o_{out}: \text{dist}(p, o_{out}) \geq \text{dist}(p, o_{in})$. The average distance of $p$ to its (at most) $k$ neighbors in $C_i$ is then

$$\text{knhDist}(p, C_i) = \frac{1}{|\text{knh}(p, C_i)|} \sum_{o \in \text{knh}(p, C_i)} \text{dist}(p, o)$$

and the average distance for a cluster $C_i$ is

$$\text{knhDist}(C_i) = \frac{1}{|C_i|} \sum_{p \in C_i} \text{knhDist}(p, C_i)$$

Based on the average $k$-neighborhood distance, we define point connectivity as follows:

**Definition 4.8 Point Connectivity.** The connectivity of a point $p$ to a cluster $C_i$ is defined as

$$\text{con}(p, C_i) = \begin{cases} 1 & \text{if } \text{knhDist}(p, C_i) < \text{knhDist}(C_i) \\ 0 & \text{if } C_i = \emptyset \\ \frac{\text{knhDist}(C_i)}{\text{knhDist}(p, C_i)} & \text{else} \end{cases}$$

The connectivity of a point to the empty set is important for determining the penalty for a missed point, which does not belong to any cluster; this will be explained in Section 4.4.1.

Figure 4.1 shows examples for connectivity scenarios for spherical clusters using $k = 3$. In the left case the points are densely and rather equally
4.3. Cluster Mapping Measure

Figure 4.2: The errors due to misplaced points (blue circles) is less severe in the right example.

distributed within the cluster. Point $p_1$ has a strong connectivity to cluster $C_1$, since its average neighborhood distance is similar to that of $C_1$. In $C_2$, the points are not equally distributed. Point $p_2$ is not strongly connected, since its average neighborhood distance is larger than the average of $C_2$. In the right example, $p_3$ also has a comparably large average neighborhood distance, but it is still strongly connected to $C_3$, because $C_3$ is less dense.

The connectivity of a point to the empty set is important for determining the penalty for a missed point, which does not belong to any cluster; this will be explained in Section 4.4.1.

Figure 4.2 illustrates the connectivity of points from the gray cluster (blue circles) to the black cluster. Misplacing gray points in the black cluster seems more severe in the left example, since they are strongly connected to their own (gray) cluster and hardly connected to the black cluster (cf. blue circles). In contrast, on the right they have a stronger connection to the black cluster making the misplacement error intuitively less severe.

4.3.3 Cluster Mapping

To decide whether a point is misplaced we need a mapping from clusters returned by a stream clustering algorithm to ground truth classes.

We base the decisions on the class distributions of clusters. We define these distributions of classes inside a cluster as frequency histograms.
An Evaluation Measure for Clustering on Data Streams

Figure 4.3: Mapping from found clusters to ground truth clusters is based on class distributions. Mapping clusters based on majority voting cannot recognize emerging or disappearing clusters.

**Definition 4.9** For a given clustering $C = \{C_1, \ldots, C_k\}$ and ground truth $CL = \{Cl_1, \ldots, Cl_l\}$, the histogram of the Class Distribution in cluster $C_i$ is

$$\rho(C_i) = (|C_i \cap Cl_1|, \ldots, |C_i \cap Cl_l|)$$

$\rho(C_i)_a$ is the $a$-th component of $\rho(C_i)$ representing the frequency of class $Cl_a$ in cluster $C_i$. (For simplicity, we omitted the noise cluster and the noise class.) The definition analogously applies to ground truth clusters $Cl_o^j$ that result from classes $Cl_j$ (cf. Def. 4.6): $\rho(Cl_o^j) = (|Cl_o^j \cap Cl_1|, \ldots, |Cl_o^j \cap Cl_l|)$.

The right part of Figure 4.3 illustrates class distributions for ground truth clusters and for found clusters.

One possibility to obtain the mapping between clusters and ground truth classes would be to use a majority voting. In that case, each cluster is mapped to the class dominating the cluster. Accordingly, for a given cluster $C$ and a
4.3. Cluster Mapping Measure

ground truth $CL = \{C_l, \ldots, C_l\}$, the cluster $C$ would be assigned to the class $\text{argmax}_{C_i \in CL} \{ \rho(C)_j \}$.

However, Figure 4.3 demonstrates that on evolving data streams a simple majority voting for mapping found clusters to ground truth clusters is not feasible. As described in Section 4.3.1, clusters $C_{l}^o$ resulting from the ground truth classes (cf. Def. 4.6) may overlap. Such an overlap scenario is illustrated in Figure 4.3(a). The gray points belong to class $C_{l_1}$, the black points to class $C_{l_2}$. The solid circles are the cluster boundaries resulting from a spherical clustering model, i.e. $C_{l_1}^o$ and $C_{l_2}^o$, and there are black points that fall into the boundary of the gray cluster and vice versa. The core problem are clusters of different densities, frequently occurring in streaming scenarios due to novelty or disappearance of clusters. In the figure, the black cluster is dense and the gray cluster is still emerging, i.e. there are only 10 objects present in the gray concept. The overlapping caused by the convex model yields a majority of the black class even in the ground truth cluster $C_{l_1}^o$ of the gray class, i.e. a majority voting would map both $C_{l_1}$ and $C_{l_1}^o$ to the wrong (black) class due to the dominance of the black objects. A majority voting cannot obtain a correct mapping in these scenarios because its decision is based on a single class’s objects in a cluster, ignoring the other classes’ information that often indicate novel or fading clusters.

To solve this issue we map clusters $C_i$ to ground truth classes $C_{l_j}$ according to the similarity of $C_i$’s and $C_{l_j}^o$’s class distributions, i.e. we take the clustering model into account. Concretely, we map cluster $C_i$ to the ground truth class $C_{l_j}$, whose ground truth cluster $C_{l_j}^o$ covers the majority of $C_i$’s class frequencies.

Definition 4.10 Cluster Mapping. For a given clustering $\mathcal{C} = \{C_1, \ldots, C_k, C_\emptyset\}$ and a ground truth $\mathcal{CL} = \{C_l, \ldots, C_l\}$,

$$\Delta(C_i, C_{l_j}^o) = \sum_{a=1}^{l} \max \{0, \rho(C_i)_a - \rho(C_{l_j}^o)_a\}$$

is the Total Surplus of objects from classes $C_{l_1}, \ldots, C_{l_l}$ in $C_i$ compared to $C_{l_j}^o$, where the surplus is the number of class objects from $C_{l_j}$ not covered by the class
distribution of the ground truth cluster $Cl^*_i$, $C_i$ is then mapped to the class

$$map(C_i) = \begin{cases} 
\arg\min_{Cl_j \in CL} \{ \Delta(C_i, Cl_j^*) \} & \text{if } \forall Cl_j : \Delta(C_i, Cl_j^*) > 0 \\
\emptyset & \text{if } C_i = C_{\emptyset} \\
\arg\max_{Cl_j \in CL} \{ |C_i \cap Cl_j^*| \} & \text{else} \\
\Delta(C_i, Cl_j^*) = 0 
\end{cases}$$

The first and most important case ensures that for a set of ground truth clusters, $C_i$ is mapped to the ground truth class that induces the least surplus on class frequencies. If we would also consider negative differences, as e.g. in $L_p$-norms, micro clusters would always be mapped to the smallest ground truth class. Employing a majority voting is not generally feasible in the streaming context as explained above. The second case ensures that the unassigned object set $C_{\emptyset}$ is not mapped to any class, enabling a correct penalty for missed non-noise points (cf. analysis in Section 4.4.1). The third case arises if $C_i$ completely fits into one or more ground truth clusters; in this special case, a majority voting is applied. If a ground truth cluster $Cl^*_i$ falls completely into another cluster $Cl^*_j$, i.e. $\forall a : \rho(Cl^*_i)_a \leq \rho(Cl^*_j)_a$, $C_i$ is mapped to the larger cluster’s class; such clusters are inseparable, as we discuss in Section 4.4.3. If we use this mapping in Figure 4.3, the found cluster $C_1$ in (b) as well as the ground truth clusters $Cl^*_i$ in (a) are correctly mapped.

### 4.3.4 Cluster Mapping Measure

With the mapping from found clusters to ground truth classes, we can now determine the set $F \subseteq O^+$ of points that cause faults, i.e. missed points, misplaced points, or included noise points.

**Definition 4.11 Fault Set.** For objects $O^+ = O \cup Cl_{\text{noise}}$, a ground truth $CL^+ = CL \cup \{ Cl_{\text{noise}} \}$, and a clustering $C = \{ C_1, \ldots, C_k, C_{\emptyset} \}$, the set of objects mapped to a false class is

$$F = \{ o \in O^+ | \exists C_i : o \in C_i \land map(C_i) \neq Cl(o) \}$$
4.3. Cluster Mapping Measure

Obviously, a single point \( o \) can cause several faults. The set of clusters in which \( o \) causes a fault is

\[
\text{faultClu} (o) = \{ C_i \in \mathcal{C} | o \in C_i \land \text{map}(C_i) \neq \text{Cl}(o) \} 
\]

The penalty for a fault point depends on the connectivity to both its true class and the assigned class.

**Definition 4.12 Penalty.** Let \( o \) be an object from the fault set and let \( C_i \in \text{faultClu}(o) \). The penalty for \( o \) is

\[
\text{pen}(o, C_i) = \text{con}(o, \text{Cl}(o)) \cdot (1 - \text{con}(o, \text{map}(C_i)))
\]

The **Overall Penalty** for a fault point \( o \) with respect to all found clusters \( C_i \in \mathcal{C} \) is defined as

\[
\text{pen}(o, \mathcal{C}) = \max_{C_i \in \text{faultClu}(o)} \{ \text{pen}(o, C_i) \}
\]

The single penalty function can handle all three fault types, and we detail on this in Section 4.4.1. To sustain comparability between points, we only count a fault point’s most serious fault in the overall penalty.

Now we can define the CMM, which indicates how different a given clustering is from a given ground truth. CMM is a normalized sum of the penalties: if no fault occurs CMM will be 1 and 0 indicates maximal error.

**Definition 4.13 CMM.** Given an object set \( \mathcal{O}^+ = \mathcal{O} \cup \text{Cl}_{\text{noise}} \), a ground truth \( \mathcal{CL}^+ = \mathcal{CL} \cup \{ \text{Cl}_{\text{noise}} \} \), a clustering \( \mathcal{C} = \{ C_1, \ldots, C_k, C_\emptyset \} \), and the fault set \( \mathcal{F} \subseteq \mathcal{O}^+ \), the **Cluster Mapping Measure** between \( \mathcal{C} \) and \( \mathcal{CL}^+ \) is defined using the point weight \( w(o) \) and the overall penalty from Definition 4.12 as

\[
\text{CMM}(\mathcal{C}, \mathcal{CL}) = 1 - \frac{\sum_{o \in \mathcal{F}} w(o) \cdot \text{pen}(o, \mathcal{C})}{\sum_{o \in \mathcal{O}^+} w(o) \cdot \text{con}(o, \text{Cl}(o))}
\]

and if \( \mathcal{F} = \emptyset \), then \( \text{CMM}(\mathcal{C}, \mathcal{CL}) = 1 \). In the definition, it is reflected that the penalty for non-fault objects is 0.
4.4 Analysis and Refinement

In this section we first analyze how CMM handles different fault types caused by evolving streams. In Section 4.4.2 we refine CMM such that it is robust against errors occurring due to the clustering model and that it can measure how well a clustering reflects the ground truth’s main concepts. Section 4.4.3 introduces a structure analysis for datasets.

4.4.1 Missed Objects, Misplaced Objects, Noise Inclusion

In this section we discuss how the penalties differ for the three fault types and why these faults are correctly reflected.

An object \( o \in \mathcal{CL} \) is missed if it is unassigned and is not a noise object, i.e. \( o \in C_{\emptyset} \wedge o \notin CL_{\text{noise}} \). The set of these objects is called \( \mathcal{F}_{\text{missed}} \). The connectivity of \( o \) to its class \( Cl(o) \) gives an idea of how severe it is if an algorithm excludes \( o \) from a cluster mapped to \( Cl(o) \). More precisely, if \( o \) is hardly connected to \( Cl(o) \) then the exclusion of \( o \) is not as severe as an exclusion of a highly connected object. This is expressed by the first term in the penalty definition. The second term equals to 1 according to Definition 4.8, i.e. \( o \in \mathcal{F}_{\text{missed}} \Rightarrow \)

\[
pen(o, C_{\emptyset}) = con(o, Cl(o)) \cdot (1 - con(o, \emptyset))
\]

If a noise object \( o_{\text{noise}} \) is assigned to a cluster \( C_i \), this noise inclusion is considered a fault, and the corresponding set is called \( \mathcal{F}_{\text{noise}} \). We want to penalize noise inclusion according to the connection between \( o_{\text{noise}} \) and \( map(C_i) \), i.e. if \( con(o_{\text{noise}}, map(C_i)) \) has a high value then we want to assign a low error and vice versa. This is expressed by the second term in the penalty definition, the first term is close to 1 due the random distribution of noise, i.e. \( o \in \mathcal{F}_{\text{noise}} \Rightarrow \)

\[
pen(o, C_i) = con(o, Cl_{\text{noise}}) \cdot (1 - con(o, map(C_i)))
\]

\( = 1 \) for noise
4.4. Analysis and Refinement

For a misplacement error, assume an object $o$ from class $Cl(o)$ is assigned to cluster $C_i$ and $C_i$ is mapped to another class, i.e. $map(C_i) \neq Cl(o)$. Misplaced objects are summarized in $F_{misplaced}$. In this case we want to consider both the connection of $o$ to its true class $Cl(o)$ and its connection to the wrongly assigned class $map(C_i)$. More precisely, a high value for $con(o, Cl(o))$ yields a high error and a low value for $con(o, map(C_i))$ yields a high error as well. The penalty combines both aspects by multiplication.

Better understanding of clusterings and their evaluation can be achieved by constraining CMM (cf. Definition 4.13) to one of the subsets $F_{missed}$, $F_{noise}$, or $F_{misplaced}$ (cf. Section 4.5).

4.4.2 Model Error and Concept Detection

If one wants to evaluate the goodness of a given clustering algorithm, the corresponding approach, i.e. its underlying clustering model, is of importance. For example, given the datasets from Figure 4.2 or 4.3, a spherical clustering algorithm can never reach a perfect result, since even the ground truth clusters (i.e., the best obtainable clusters according to the spherical model; cf. Definition 4.6) contain objects from different classes yielding misplacement errors in CMM. How meaningful is a measure telling you an algorithm has 70% performance if you don’t know whether its model even allows it to perform better? In the following, we the modify CMM such that perfect results ($CMM \rightarrow 1$) are achieved for ground truth clusterings. In case of the evaluation of ground truth clusterings, the mapping component of CMM is not needed: each ground truth cluster $Cl_j$ is mapped to its corresponding class, i.e., $map(Cl_j) = Cl_j$.

CMM shall measure how well an algorithm solved the clustering task with respect to the spherical model. To this end it is checked for each object, whether it causes an error in the corresponding ground truth clusters $Cl_j$. If such an error exists, it is called an error by model and the object is added to the set $F_{model}$. Formally, $o \in F_{model} \iff |\{Cl_i | Cl_i \in CL \land o \notin Cl_i \land o \in Cl_j\}| > 0$. These errors by model are then excluded in the CMM computation. This way the ground truth always yields a perfect result and so can an algorithm.
Any stream clustering algorithm has to adhere to memory constraints for its internal representation, i.e. the total number of clusters or micro clusters is limited. Due to aging of objects, drifting clusters, and newly emerging clusters the algorithm constantly has to update its representation and decide on the position and size of its clusters. It is more important to reflect the main concepts of the current data distribution rather than spending limited resources to represent some concepts very detailed and missing other concepts in exchange. In other words, for two given clusterings with equally many missed objects, the one that better covers the concepts is preferable (cf. Figure 4.4).

To account for this aspect we decrease the penalty for missed points with respect to the severity of the fault, i.e., we reduce the penalty if a missed point is nearly captured by a cluster mapped to the object’s class. Formally, we redefine the overall penalty for this fault type. For an object \( o \in F_{\text{missed}} \) and all clusters \( C_i \) with \( \text{map}(C_i) = Cl(o) \), the relative distance of \( o \) to the cluster boundary of \( C_i \) is analyzed: \( o \in F_{\text{missed}} \Rightarrow \)

\[
\text{pen}(o, C) = \text{con}(o, Cl(o)) \cdot \min_{C_i \in C \atop \text{map}(C_i) = Cl(o)} \left\{ 1 - e^{-\frac{\min\text{Dist}(o, C_i)}{\max\text{Dist}(o, C_i)}} \right\}
\]

Accordingly, the smaller the ratio \( \frac{\min\text{Dist}}{\max\text{Dist}} \), the smaller the penalty. An object’s penalty is minimized over all clusters that are mapped to the object’s class, i.e., it suffices that one cluster nearly covers the object. The second term simplifies to \( 1 - e^{-\frac{d}{2r}} \) for spherical clusters, where \( d \) is the distance of \( o \) to \( C_i \)’s center and \( r \) its radius.
4.4.3 Structure Analysis

To analyze the structure of a given dataset, we define the connectivity between two classes, which is based on the same concept as Definition 4.8. Intuitively, the connection from class $C_l$ to class $C_j$ is high, if several points that have a strong connection within $C_l$ show a strong connection to $C_j$ at the same time. The definition below uses the same locality parameter $k$ as Definition 4.7.

**Definition 4.14 Class connectivity.** For a locality parameter $k$ the connectivity from class $C_l$ to class $C_j$ is

$$\text{con}_{C_l \rightarrow C_j} = \max_{\{KM \subseteq C_l : |KM|=k\}} \left\{ \frac{1}{k} \sum_{p \in KM} \text{con}(p, C_l) \cdot \text{con}(p, C_j) \right\}.$$

The symmetric connectivity between two classes is then

$$\text{con}(C_l, C_j) = \min\{\text{con}_{C_l \rightarrow C_j}, \text{con}_{C_j \rightarrow C_l}\}.$$

The above definition considers the $k$ points within $C_l$ that maximize the product of connectivity to both classes. The blue objects in Figure 4.2 illustrate the connectivity between the two classes gray and black. In both cases the two classes are not strongly connected.

If two ground truth classes have a strong connectivity, the two classes are difficult to separate from each other. To avoid misplacement penalties between such classes, we merge classes who’s connectivity according to Definition 4.14 exceeds a threshold $\tau$ and assign their points the same label. Analyzing the structure of a given ground truth $\mathcal{CL} = \{C_l_1, \ldots, C_l_l\}$ hence yields a reduced set of classes $\mathcal{CL}^\prime = \{C_l'_1, \ldots, C_l'_h\}$ with $h \leq l$. As an indicator of how well the classes in a given ground truth $\mathcal{CL}^+$ are separable we define:

$$\text{Separability}(\mathcal{CL}^+) = \frac{h}{l}.$$

In Figure 4.5 we demonstrate for different horizons how the Separability is influenced by different $\tau$ values.
For the structure analysis in the experiments in Section 4.5, we use \( \tau = 0.5 \). To complete the structure analysis we define a second indicator, counting the percentage of objects represented redundantly, i.e. covered by several ground truth clusters \( Cl^o_i \):

\[
Red(CL^+) = \frac{|\{o \in O^+ | o \in Cl^o_i \wedge o \in Cl^o_j \wedge i \neq j\}|}{|O^+|}
\]

This indicator allows for a more in-depth analysis of the different measures’ behavior in the empirical evaluation.

### 4.5 Experiments

We evaluate CMM in comparison to evaluation measures introduced in Section 4.2. For repeatability, we integrated CMM into MOA (cf. Section 4.6). First of all, we describe our evaluation framework.

**Stream generator.** The generator’s output is a stream of data points that have class labels and weights reflecting a point’s age, depending on the used horizon. Since clusters move, a larger horizon yields more tail-like clusters; a small horizon yields more sphere-like clusters. Technically, the data points are obtained by generating clusters, which have a center and a maximal radius and which move through a unit cube \(([0, 1]^d)\) into a random direction. At the cube boundaries, the cluster center bounces off (and no points are generated that are positioned outside the unit cube). At specific intervals all cluster centers are shifted by 0.01 in their corresponding direction and points
are equally drawn from each generating cluster. On average, the number of points in each cluster is the same. Our stream generator has the following parameters: the number of clusters moving through the dataspace, the number of generated points, the shift interval, the cluster radius, the dimensionality, and the noise degree.

**Real world data streams.** We use the 10 continuous attributes of *Forest Cover Type*†, consisting of 581,012 objects and 7 classes, and we use the 34 continuous attributes of *Network Intrusion*‡, consisting of 4,898,431 objects and 5 classes. This variant was also used in [AHWY03, CEQZ06].

**Cluster generator.** We evaluate measures in specific clustering error scenarios, which are obtained by generating clusterings out of the synthetic stream that reflect a desired error level. We create cluster join, radius decrease, and cluster remove errors and the cluster boundaries are determined by a technique from [Gar99]. The cluster join error simulates that two classes are (incorrectly) covered by only a single cluster. Technically, these errors are obtained by joining pairs of nearby, non-overlapping clusters; e.g., an error level of 0.4 indicates that all pairwise clusters are joint whose minimal boundary distance is below 40% of the minimum of their cluster radii. By this, we achieve a more realistic scenario without joining of far apart clusters. The radius decrease error states that the generated clusters have a radius that is smaller than the radius of the corresponding ground truth cluster. An error level of 1 states that the radius of the error cluster is 0; for an error level of 0 the two radii are equal. The cluster remove error denotes how many clusters are missed by a clustering; for an error level of 0.4, 40% of the clusters are removed from the error-free clustering.

**Setup.** We compare CMM and its variants (cf. Section 4.4.1) with internal and external measures from the literature (cf. Section 4.2). In selected experiments, we provide the structural measures Separability and Redundancy (cf. Section 4.4.3). All measures, besides the Sum of Squared distances (SSQ), are normalized to [0, 1]. For some measures the best value corresponds to 0; we reverse them and indicate this by ”1-“ in the plots. The values of SSQ

---

†http://archive.ics.uci.edu/ml/datasets/Covertype
Figure 4.6: 10-dimensional Forest Cover Type stream: (a): Illustration of the individual evaluations for a real stream clustering and the resulting medians. (b-d): Varying horizon on ground truth clusterings. are plotted on a secondary y-axis. By using clusterings from the cluster generator, we ensure an evaluation unbiased by possible incorrect outcomes of stream clustering algorithms. Moreover, we evaluate real clusterings of CluStream [AHWY03] and DenStream [CEQZ06] using CMM. The CMM parameter $k$ was preliminary fixed to 2; the choice of $k$ only has marginal influence on CMM effectiveness, as we will later illustrate. Synthetic data is generated with the following settings: the dimensionality is 2, the number of points is 200,000, the cluster number is 6, the cluster radius is 0.075, the shift interval is 100 points, and noise is equally distributed and fixed at 10%.

In the experiments, the horizon (cf. Definition 4.3) is defined by the number of included points, i.e. $h = |H|$, and the evaluation frequency is the horizon. The values in all plots are the medians over all evaluations of the ana-
4.5. Experiments

Figure 4.7: CMM and internal measures under different error types with horizon $h = 5,000$.

Figure 4.6 shows experiments where we analyze the robustness of the measures under a varying horizon. We use error-free Forest Cover Type clusterings, i.e., the ground truth. By a varying horizon, this ground truth changes; with greater horizons, the cluster tails become longer, increasing the probability of overlapping clusters. This is confirmed by the two structural measures in Figure 4.6(b): Redundancy rises while Separability (of clusters) drops. The comparison of CMM with the competing measures shows a clear picture: CMM (in Figure 4.6(b)) achieves perfect quality that is stable over all settings. In contrast, both internal (Figure 4.6(c)) and external measures (Figure 4.6(d)) degrade with increasing horizons, showing that they cannot cope with higher redundancy and the worsened separability that occurs with larger horizons. More surprisingly,
Error level. We analyze how good the measures can reflect the errors in generated clusterings. We start with the internal measures in Figure 4.7, where each graph corresponds to one error type. Comparing an external measure like CMM with internal measures is inherently unfair; however, since internal measures are often used in practice, a general overview of their performance in error scenarios is of interest. From the three plots, we can conclude the following: The C Index shows nearly no reaction; only extreme cases (error $\rightarrow 1$) for radius decrease and cluster remove errors are indicated through quality drops. The Silhouette Coefficient reflects the cluster join error, but cannot reflect the other errors, where the corresponding values even rise. The SSQ, which is used in many stream clustering publications [AHWY03, CT07], shows a good reaction on the cluster join and the cluster remove error; however, if we look at the corresponding maximal val-
ues (60 and 10,000) a practical use is questionable due to its unnormalized nature. CMM shows a very good performance on all scenarios, and a deeper analysis will be included in the next paragraph.

In the following, we compare CMM with the external measures using horizons of 5,000 and 10,000. We start with cluster join errors in Figure 4.8. Due to the technique used for creating this error (only nearby clusters are joined), the error level 1 does not indicate that all clusters are joined into a single cluster. Figure 4.8(a) and 4.8(b) show that CMM reflects this error well, i.e. with an increasing number of joined clusters the quality constantly drops. We expect this to be caused by misplaced points, i.e. points that are assigned to wrong clusters, which is confirmed by CMM\textsubscript{misplaced}. CMM\textsubscript{noise} illustrates that when clusters are joined the resulting clusters tend to absorb noise points due to their larger sizes. The external measures in Figure 4.8(c) also reflect the errors. However, a general problem of these measures are error-free settings (error=0) in which they show lower qualities (0.8-0.6).
An Evaluation Measure for Clustering on Data Streams

Figure 4.10: Evaluating CMM and external measures: cluster remove errors.

We suspect this to be caused by cluster overlap and it worsens with a larger horizon, as illustrated in Figure 4.8(d). Considering that these measures are normalized to $[0, 1]$, this is a severe problem. In contrast, CMM shows maximal quality in error-free settings for both horizons and this also holds for the other error types. The cluster radius decrease error is analyzed for CMM in Figure 4.9(a) and 4.9(b). This error mostly results in missed points, which is shown by CMM$_{missed}$ and thus by CMM. Here we also demonstrate the influence of the CMM refinement that better measures how well the main concepts of the data are reflected by a clustering (cf. Section 4.4.2). The figure shows that at lower error levels, CMM does not decrease as rapidly as the variants without refinement (green lines). This is intended, since the main concepts of the data are well represented at this stage and only close points are missed. From the six measures in Figure 4.9(c), four can basically reflect the error; however, for lower error levels ($\leq 0.4$) three of the measures go up and the negative effects of larger horizons on all competing measures
4.5. Experiments

Figure 4.11: Marginal effects (standard deviation $< 0.009$) of varying $k$.

are again obvious in Figure 4.9(d). Especially the bad performance of Purity is mentionable, a measure which is the main evaluation method in many stream clustering publications [CEQZ06, JZC06]. The cluster remove error is analyzed in Figure 4.10(a)-4.10(d). This error corresponds to removal of complete clusters and thus concepts; accordingly and in contrast to the radius decrease error this loss of concepts should be reflected by a constant decrease of the measured quality. This is achieved by CMM. A comparison to its variant without refinement shows that for this error type the refinement has no undesired effects; this refinement only affects the measurement of radius decrease errors. The external measures show a similar behavior as for the radius decrease error: only three reflect the error at all and their expressiveness strongly decreases for larger horizons.

Summarized, CMM can precisely reflect all errors types. Most of the internal measures cannot reflect the errors at all, while some of the external measures reflect the errors but have serious problems in settings without errors or larger horizons.

CMM locality parameter $k$. In Figure 4.11 we analyze the influence of $k$ (cf. Section 4.3.2). We simulate a more realistic evaluation scenario by using synthetic clusterings with embedded errors (error level=0.5). For each error type, we tried 10 different $k$ ($k \in \{1, \ldots, 10\}$) with 4 different horizons. On the left we plot the standard deviation of the CMM results with respect to the choice of $k$: CMM values for two of the error types are influenced by $k$ for very small horizons; however, considering the value range of the CMM and the small standard deviations ($< 0.009$), these effects are negligible. This is also illustrated on the right, where we plot the exact CMM values with a
horizon of 10,000 for the individual $k$ on the range $[0.8, 0.98]$. Both figures show that $k$ has only marginal influence for very small horizons and does not affect CMM for larger horizons.

**Real data and stream clustering algorithms.** Finally, we analyze the measures in real application scenarios. We cluster real world data streams by actual stream clustering algorithms, in this case CluStream [AHWY03] and DenStream [CEQZ06]. In Figure 4.12, we distinguish between macro clusterings, i.e. the output of the algorithms, and micro clusterings, i.e. the underlying internal representations constantly adapted by the algorithms. The macro clusterings are produced from the micro clusterings by an offline component. Comparability is ensured by using CluStream’s offline component in CluStream and DenStream. We apply an horizon of 1,000. A structure analysis of the real world streams is shown in Figure 4.13: Forest Cover
Figure 4.13: Structure analysis of real world streams. The definitions of Separability and Redundancy can be found in Section 4.4.3.

Type has high redundancy and low separability; Network Intrusion has no redundancy and high separability, making the clusters easier to find.

We start with Network Intrusion. The results obtained by DenStream for the macro clustering in Fig 4.12(a) confirm the conclusions from the structural analysis: nearly all evaluation measures indicate that high quality clusterings are generated. For CluStream, however, we can make a surprising conclusion. CMM and most of the other measures indicate that the found clusterings are of low quality. This is mostly caused by missed points, as CMM_Missed shows. In contrast, the evaluation of micro clusterings used by CluStream in Figure 4.12(c) shows another picture: the micro clusterings are of high quality. We can conclude that CluStream correctly identifies the concepts in the data, but the used offline method that generates the macro clusters from the micro clusters is ineffective for this data type.

For Forest Cover Type in Figure 4.12(b) and 4.12(d), we can conclude by CMM_Missed that DenStream covers nearly all of the concepts in the macro clusterings. CluStream, however, seems to miss more concepts of the data. For this dataset, the macro clusterings of both CluStream and DenStream are of higher quality than the micro clusterings.

Summarized, CMM is an effective measure for evaluation on real data streams and for clusterings generated by stream clustering algorithms, and the CMM variants help in understanding why an analyzed clustering is of a specific quality.
4.6 Stream Clustering Evaluation in MOA

In this section, we present our experimental framework in which we integrated the CMM. The framework offers the means for extensive evaluation and visualization of stream clusterings. It is an extension of the Massive Online Analysis (MOA) software environment released under GNU GPL License.

MOA [BHP+10] was recently introduced and builds on the Waikato Environment for Knowledge Analysis (WEKA) framework [HFH+09], which was developed at the University of Waikato, New Zealand. So far, MOA only considered stream classification algorithms. Accordingly, no stream clustering evaluation tool existed offering a suite of implemented stream clustering algorithms and evaluation measures.

Within MOA, we build an experimental framework for clustering data streams, making it easy for researchers to run experimental data stream benchmarks. The MOA framework offers such possibilities for classification algorithms on data streams. Our extension of MOA to stream clustering offers the following new features:

- data generators for simulation of cluster evolution (including events like emerging and merging clusters [SNTS06]),
- an extensible set of stream clustering algorithms,
- an extensible set of evaluation measures including the CMM,
- methods to assess evaluation measures under specific error scenarios,
- visualization for analyzing results and comparing different settings.

The new version of MOA contains several stream clustering methods such as CluStream [AHWY03], ClusTree [KABS09], StreamKM++ [AMR+12], D-Stream [CT07], DenStream [CEQZ06], and CobWeb [Fis87]. It contains measures for analyzing the performance of the clustering models generated.

§An article on MOA has been published in the Journal of Machine Learning Research (JMLR) [BHP+10] and MOA has been presented at different international conferences including IEEE ICDM 2010 [KKJ+10].
from both online and offline components. The visualization component allows to visualize the stream as well as the clustering results, choose dimensions for multidimensional settings, and compare experiments with different settings in parallel.

Figure 4.14(left) shows the configuration dialog for our data generator with events. Generally, dimensionality, number, and size of clusters can be set as well as the drift speed, decay horizon (aging), noise rate, etc. Events constitute changes in the underlying data model such as growing of clusters, merging of clusters, or creation of new clusters [SNTS06]. Using the event frequency and the individual event weights, one can study the behavior and performance of different approaches on various settings.

Figure 4.14(right) shows our visualization tab. For this screenshot two different settings of the CluStream algorithm [AHWy03] were compared on the same stream setting (including merge/split events every 50,000 examples) and four measures were chosen for online evaluation (F1, Precision, Recall, and SSQ). The upper part of the GUI offers options to pause and resume the stream, adjust the visualization speed, choose the dimensions for x and y as well as the components to be displayed (points, micro- and macro clustering, ground truth). The lower part of the GUI displays the measured values for both settings as numbers (left side, including mean values) and
the currently selected measure as a plot over the arrived examples (right, F1 measure in this example).

4.7 Conclusion

Evaluation of clustering results on static datasets received a lot of research attention for several decades. Many recent publications deal with the important task of clustering on evolving data streams. However, despite the amount of work on evaluation measures and on stream clustering algorithms, no effort has been made to meet the requirements of both tasks. In this chapter we proposed a novel and effective evaluation measure for clustering on evolving data streams called CMM. It is the first measure that takes the important properties of the streaming context into account. It is based on a novel mapping component that can handle emerging and disappearing clusters correctly. We included CMM into the open source MOA framework [MOA] and showed in extensive experiments on real and synthetic data that it is a robust measure precisely reflecting errors in data stream scenarios.

The core of CMM is the novel mapping component, which compares class distributions of clusters. Another kind of data mining task applying cluster mapping is cluster tracing, in which evolving clusters are tracked over time. So far, all applied mapping techniques, as well as the CMM, rely on individual object identifiers in the mapping process. In the next chapter, we develop an approach for cluster tracing that applies a novel paradigm for cluster mapping. In this paradigm, clusters are mapped by the similarity of their characteristics, independently of individual object identities.
Chapter 5

Tracing of Evolving Subspace Clusters

∗Cluster tracing algorithms mine temporal evolutions of clusters. Generally, clusters represent groups of objects with similar values. In a temporal context like tracing, similar values correspond to similar behavior in one snapshot in time. Each cluster can be interpreted as a behavior type and cluster tracing corresponds to tracking similar behaviors over time. Existing tracing approaches are for datasets satisfying two specific conditions: The clusters appear in all attributes, i.e. fullspace clusters, and the data objects have unique identifiers. These identifiers are used for tracking clusters by measuring the number of objects two clusters have in common, i.e. clusters are traced based on similar object sets.

These conditions, however, are strict: First, in complex data, clusters are often hidden in individual subsets of the dimensions. Second, mapping clusters based on similar objects sets does not reflect the idea of tracing similar behavior types over time, because similar behavior can even be represented by clusters having no objects in common. A tracing method based on similar object values is needed. In this chapter, we introduce a novel approach that traces subspace clusters based on object value similarity. Neither subspace tracing nor tracing by object value similarity has been investigated before.

∗This chapter has been published in the Data Mining and Knowledge Discovery Journal (DMKD) [KGS10]. A shorter version was published in the Proceedings of the 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2011) [GKLS11a].
5.1 Motivation

Temporal properties of patterns and their analysis are under active research [BHS08]. A well known type of pattern are clusters, which correspond to similarity-based groupings of data objects. A good example for clusters are customer groups. Clusters can change in the course of time and understanding this evolution can be used to guide future decisions [BHS08], e.g. predicting whether a specific buying behavior will occur. Another example is climate analysis: understanding of pattern evolution could be used in the development of methods to prevent further climate change. Cluster evolutions are mined by tracing approaches that find mappings between clusters of consecutive time steps [KMB05, RG08, SNTS06]. These mappings describe the cluster evolutions over time.

Cluster tracing goes beyond time series clustering [Fu11, Lia05] by allowing objects to be assigned to different clusters over time. Nevertheless, the existing algorithms for cluster tracing have a severe limitation: Clusters are mapped if the corresponding object sets are similar, i.e. the algorithms check whether the possibly matching clusters have a certain fraction of objects in common; they are unable to map clusters of different objects, even if the objects have similar attribute values. However, in domains as for example climate data analysis, we are only interested in clusters representing general phenomena and not in clusters of specific individual objects.

In this chapter, we introduce a novel tracing method to map clusters solely based on their characteristics; more concretely, the individual object sets of clusters are not the deciding factor for the mappings. In our approach, two clusters at different points in time can show the same characteristics, but the underlying cluster objects can be completely different. We term this object-value-based tracing. This new paradigm is relevant, as the following examples illustrate.

Consider the relationship between the attributes temperature and a specific biomass in the oceans, a relationship which can be represented by clusters. For example that a specific temperature occurs together with a specific amount of biomass. The analyzed data is captured by sensors that are po-
Motivation

Positioned at fixed grid cells in the ocean or the data is the result of complex simulations. It is obvious that the detected clusters representing the relationship evolve and that the observed values are recorded at different grid cells as time progresses. Clusters found by time series clustering approaches or sensor-identity-based tracing cannot express the evolutions. Another example is the buying behavior of people in different countries. Often it is similar, but shifted in time. For example, the buying behavior in Europe is similar to the behavior in North America, but only some months later. A cluster tracing algorithm should detect the presented phenomena. However, existing methods do not, since the observed populations stay at the same place, and thus there are no shared objects between clusters — only the behavior migrates.

With today’s complex data, patterns are often hidden in different subsets of the dimensions; for detecting these clusters with locally relevant dimensions, subspace clustering was introduced. However, despite that many temporal datasets are of this kind, e.g. gridded scientific data, subspace clustering has never been used in a cluster tracing scenario. The existing cluster tracing methods can only cope with fullspace clusters, and thus cannot exploit the information mined by subspace clustering algorithms. Our novel tracing method measures the subspace similarity of clusters and thus handles subspace clusters by design.

Summarized, we introduce a method for tracing behavior types in temporal data; the types are represented by clusters. The decision, which clusters of consecutive time steps are mapped is based on a novel distance function that tackles the challenges of object value similarity and subspace similarity. Our approach can handle the following developments: emerging or disappearing behavior as well as distinct behaviors that merge into uniform behavior and uniform behavior that develops into several distinct behaviors. By using subspaces, we enable the following evolutions: Behavior can gain or lose characteristics; i.e., the representing subspace clusters can gain or lose dimensions over time, and clusters that have different relevant dimensions can be similar. Varying behavior can be detected; that is, to some extent the values of the representing clusters can change.
Figure 5.1: Top: three consecutive time steps, with each time step corresponding to a different database. (We used varying object symbols to illustrate that the time steps do not share objects.) Bottom: possible clusterings with respect to a cell-based clustering paradigm (lower and upper bounds in relevant dimensions) and exemplary cluster tracings.

Figure 5.1 exemplifies the evolution of temperature and biomass measurements for three consecutive time steps. The upper part of the figure shows the objects; the lower part abstracts from the objects and illustrates possible clusterings of the databases and tracings between the corresponding clusters. The three time steps do not share objects, i.e. each time step corresponds to a different database; to illustrate this, we used varying object symbols. A cell-based clustering paradigm is assumed, i.e. clusters are defined by lower and upper bounds in each of the two dimensions, and in case of subspace clusters, bounds are only given for the relevant dimensions. For example, the cluster $C_{1,2}$ at time step $t = 1$ is a fullspace cluster in which the temperature and biomass measurements of the cluster's objects are constrained to specific intervals; for the subspace cluster $C_{1,1}$, only the temperature values are constrained to an interval, while the biomass measures are scattered over the whole dimensional extent. An example for behavior that gains characteristics is the mapping of cluster $C_{1,1}$ to $C_{2,1}$: at time step $t = 1$ only the biomass measurements are positioned in a specific interval, while at
Comparison with Related Work

$t = 2$ also the temperature measurements can be constrained by an interval, i.e. this cluster gains one dimension. Varying behavior is illustrated by the mapping from $C_{1,2}$ to $C_{2,2}$; the values of the cluster have changed. If the two dimensions of the databases were spatial, this could be interpreted as a movement. A behavior split-up can be seen from time step $t = 2$ to $t = 3$: the single cluster $C_{2,1}$ is mapped to the two clusters $C_{3,1}$ and $C_{3,2}$.

Summarized, our contributions are:

- We introduce a novel tracing approach for evolving subspace clusters in high dimensional spatio-temporal data. Subspace clusters are traced based on similar object values and not based on object sets.

- We distinguish several kinds of behavior developments, i.e. clusters can emerge, disappear, split, or merge.

- Degree of evolution between two subspace clusters is measured by a distance function based on subspace similarity and value similarity.

- We propose a method for information transfer between time steps, to avoid unstable clusterings. Tracing effectiveness is therefore improved, as it depends on the input clusterings.

This chapter is structured as follows: Section 5.2 discusses the related work. Section 5.3 introduces our new model for tracing subspace clusters. The effectiveness is shown in Section 5.4 and an application scenario with oceanographic grid data is discussed in Section 5.5. Section 5.6 contains concluding remarks.

5.2 Comparison with Related Work

Several temporal aspects of data are regarded in the literature [BHS08]. In stream clustering scenarios, clusters are adapted to reflect changes in the observed data (cf. Chapter 4). Stream clustering was extended to high dimensional data in [AHWY04, NZP+12]. A special case of stream clustering is for moving objects [JLO07, LHY04], focusing on spatial attributes. Stream
clustering in general, however, gives no information about the actual cluster evolution over time [BHS08]. For this, cluster tracing algorithms were introduced [KMB05, RG08, SNTS06]; they rely on mapping clusters of consecutive time steps, and the mapping decision is based on shared objects. For example, in [SNTS06] clusters are mapped if their overlap (number of shared objects) is above a specific threshold. Our approach, in contrast, maps clusters only if their corresponding object values are similar.

Time series clustering (cf. Chapter 3) and trajectory clustering [GS99, VKG02] can be seen as even more limited variants of similar object set based tracing, since the obtained clusters have constant object sets that do not change over time. Accordingly, these methods search for groups of objects that have a similar behavior over the whole time extent. There is no possibility of detecting that a behavior reflected by one time series cluster is occurring in a different cluster after some time.

Approaches for analyzing climate patterns in multivariate data such as [HHIO05] do not actually track clusters; they cluster all points in time simultaneously, neglecting the temporal information. Afterwards, the resulting clusters are remapped to their temporal extension. By this, however, highly evolving patterns get lost, because they are not considered as one cluster.

The work in [Agg05] analyzes multidimensional temporal data based on dense regions that can be interpreted as clusters. The approach is designed to detect substantial changes of dense regions; however, tracing of evolving clusters that slightly change their position or subspace is not possible.

The research area of clustering evaluation (cf. Chapter 4) is related since clusters of two given clusterings need to be mapped. These approaches are designed for another purpose and there is no consideration of several time steps. More importantly, external measures like the CMM introduced in Chapter 4 are based on object identifiers.

A further limitation of existing cluster tracing algorithms is that they do only cope with full-space clusters. Until now, subspace clusters were only applied in streaming scenarios [AHWY04], but never in cluster tracing scenarios; deciding whether subspace clusters of varying dimensionalities are similar is challenging. Our algorithm is designed for this purpose.
5.3 Model for Subspace Cluster Tracing

Our main objective is to trace behavior types and their developments over time. This is formalized in the following section. First, some basic notations: For each time step $t \in \{1, \ldots, T\}$ of our spatio-temporal data we have a $D$-dimensional database $DB_t \subseteq \mathbb{R}^D$. We assume the data to be normalized between $[0, 1]$. A subspace cluster $C_{t,i} = (O_{t,i}, S_{t,i})$ at time step $t$ is a set of objects $O_{t,i} \subseteq DB_t$ along with a set of relevant dimensions $S_{t,i} \subseteq \{1, \ldots, D\}$. The set of all subspace clusters $\{C_{t,1}, \ldots, C_{t,k}\}$ of the same time step $t$ is denoted as subspace clustering $\text{Clust}_t$. Our approach is independent of a specific clustering model. For illustration, we assume a cell-based clustering paradigm in the following, i.e. clusters are described by intervals (lower and upper bounds) in the relevant dimensions.

Each subspace cluster of our clusterings represents a behavior type. In our cell-based clustering paradigm, this could for example be a set of sensor readings expressing the relationship between biomass and temperature: the temperatures and biomass measured by many sensors are contained in a specific interval. The terms behavior, behavior type, and cluster are used interchangeably. In Section 5.3.1 we introduce how to trace a behavior over time and which temporal developments are possible. For tracing it is necessary to measure the similarity between behavior types; the formalization of this step is presented in Section 5.3.2. In Section 5.3.3 the clustering process in the single time steps is improved by incorporating temporal information, avoiding unstable clusterings and achieving higher quality clusterings. We conclude in Section 5.3.4 with a complexity analysis of our method.

5.3.1 Tracing of Behavior Types

In this section, we are interested in whether a typical behavior in time step $t$ continues in $t + 1$. In oceanographic data, for example, a cluster detected in a set of sensor readings at one time step can be rediscovered (i.e., having similar values) in the next time step in a different set of sensor readings, and these sensor readings can be obtained by a different set of sensors. Other
kinds of temporal developments are the disappearance of a behavior or a split-up into different behaviors. We have to identify these temporal developments for reasonable tracing of behaviors. Formally, we need a mapping function that maps each cluster at a given time step to a set of clusters in the next time step; these successors are called temporal continuations. Two clusters $C_{t,i}$ and $C_{t+1,j}$ are mapped if they are identified as similar behaviors. We use a distance function for clusters to measure these similarities. If the distance is small enough the mapping is performed.

**Definition 5.1 Mapping Function.** Given a distance function $\text{dist}$ for two subspace clusters, the mapping function $M_t : Clus_t \to \mathcal{P}(Clus_{t+1})$ that maps a cluster to its temporal continuations is defined by

$$M_t(C_{t,i}) = \{C_{t+1,j} \mid \text{dist}(C_{t,i}, C_{t+1,j}) < \tau\}$$

A cluster can be mapped to zero, one, or several clusters (1:n). We can map several clusters to the same cluster (m:1). We do not enforce (1:1) mappings. These properties are needed, so that disappearance or merging of behaviors can be detected. We describe all pairs of mapped clusters between two consecutive time steps by a binary relation:

$$R_t = \{(C_{t,i}, C_{t+1,j}) \mid C_{t+1,j} \in M_t(C_{t,i})\} \subseteq Clus_t \times Clus_{t+1}$$

Each tuple corresponds to one cluster mapping, i.e. for a behavior type in $t$ we have identified a similar one in the next time step $t+1$. These mappings as well as the corresponding clusters can be represented by a mapping graph. Reconsider that it is possible to map a behavior to several behaviors in the next time step (cf. Figure 5.1, $t = 2 \rightarrow t = 3$). All these behaviors, however, are not equally similar to the original behavior. We represent this by using edge weights within the mapping graph; the weights indicate the strength of the temporal continuation. We measure similarity based on distances, and therefore small weights denote a strong continuation while high weights reflect a weaker continuation.
Definition 5.2 Mapping Graph. A mapping graph $G = (V, E, w)$ is a directed and weighted graph with the following properties:

- Nodes represent clusters, i.e. $V = \bigcup_{t=1}^{T} Clus_t$
- Edges represent cluster mappings, i.e. $E = \bigcup_{t=1}^{T-1} R_t$
- Edge weights indicate the strength of the temporal continuations, i.e. $orall (C_i, C_j) \in E : w(C_i, C_j) = \text{dist}(C_i, C_j)$

In Figure 5.2 an mapping graph with edge weights is examplified. A mapping graph allows the categorization of different kinds of temporal developments.

Definition 5.3 Kinds of Temporal Developments. Given a mapping graph $G = (V, E, w)$, the behaviors represented by clusters $C \in V$ can be categorized:

- a behavior disappears, if $\text{outdegree}(C) = 0$
- a behavior emerges, if $\text{indegree}(C) = 0$
- a behavior splits up, if $\text{outdegree}(C) > 1$
- different behaviors merge to a single behavior, if $\text{indegree}(C) > 1$

In Figure 5.2 for example, the cluster $C_{3,1}$ corresponds to a disappearing behavior and the cluster $C_{4,1}$ is an emerging one. While cluster $C_{2,1}$ develops into several different behaviors, cluster $C_{4,2}$ is the result of behavior merging.
The kinds of temporal developments show whether a behavior appears in similar ways in the subsequent time step. However, it is also important to trace a behavior over several time steps. It should be noted that the characteristics of a behavior can naturally change over this time period. Thus, we denote the tracing of a single behavior over a specific period as an evolving cluster. Formally, an evolving cluster is described by a single path through the mapping graph. Based on the mapping graph in Figure 5.2 we are able to trace the evolving cluster \( C_{1,1} \rightarrow C_{2,1} \rightarrow C_{3,2} \rightarrow C_{4,2} \).

To ensure that the evolving clusters are correctly identified, we have to account for several evolution criteria to be included in our distance function. These criteria are presented in the following section.

### 5.3.2 Cluster Distance Measure

Our objective is to identify similar behaviors. Technically, a distance measure is needed to formally determine the similarity of two given clusters. Keep in mind, that measuring the similarity based on the fraction of shared objects is not meaningful in our approach. Even totally different populations can show up with a similar behavior in consecutive time steps.

We distinguish two kinds of evolution: First, a cluster can gain or lose characteristics, i.e. the relevant dimensions of a subspace cluster can evolve. Second, within the relevant dimensions the values can change over time. Both aspects have to be considered by our distance function for effective similarity measurement of evolving clusters.

**Similarity based on subspaces.** Each cluster represents a behavior type, and because we are considering subspace clusters, the characteristics of a behavior are restricted to a subset of the dimensions. If a behavior remains stable over time, its subspace remains also unchanged. The relevant dimensions of the underlying clusters are identical. Let us consider the clusters \( C_{t,i} = (O_{t,i}, S_{t,i}) \) and \( C_{t+1,j} = (O_{t+1,j}, S_{t+1,j}) \) of time steps \( t \) and \( t + 1 \). The represented behaviors are very similar if the dimensions \( S_{t,i} \) are also included in dimensions \( S_{t+1,j} \).
On the other hand, it is possible that a behavior loses some of its characteristics over time. In Figure 5.1, for example, the attribute biomass is no longer relevant in time step \( t = 3 \) for the behavior depicted on the bottom. Accordingly, a distance measure is meaningful if behavior types are rated as similar, even if they lose some relevant dimensions. That is, the smaller the term \( 1 - \frac{|S_{t,i} \cap S_{t+1,j}|}{|S_{t,i}|} \), the more similar are the clusters.

This formula alone, however, would prevent an information gain: If a cluster \( C_{t,i} \) evolves to \( C_{t+1,j} \) by spanning more relevant dimensions, this would not be assessed positively. We would get the same distance for a cluster with the same shared dimensions like \( C_{t,i} \), but without additional relevant dimensions like \( C_{t+1,j} \). Since more dimensions mean more information, we do consider this. Consequently, the smaller the term \( 1 - \frac{|S_{t+1,j} \setminus S_{t,i}|}{|S_{t+1,j}|} \), the more new information is obtained.

Usually it is more important for tracing that we retain relevant dimensions. Few shared dimensions and many new ones normally do not indicate similar behavior. Thus, we need a trade-off between retained dimensions and new (gained) dimensions. This is achieved by a linear combination of the two introduced terms:

**Definition 5.4 Distance w.r.t. Subspaces.** The similarity w.r.t. subspaces between two clusters \( C_{t,i} = (O_{t,i}, S_{t,i}) \) and \( C_{t+1,j} = (O_{t+1,j}, S_{t+1,j}) \) is defined by

\[
S(C_{t,i}, C_{t+1,j}) = \alpha \cdot \left( 1 - \frac{|S_{t,i} \cap S_{t+1,j}|}{|S_{t,i}|} \right) + (1 - \alpha) \cdot \left( 1 - \frac{|S_{t+1,j} \setminus S_{t,i}|}{|S_{t+1,j}|} \right)
\]

with trade-off factor \( \alpha \in [0,1] \). In this definition, only the sets of relevant dimensions \( S_{t,i} \) are compared, ignoring the object sets \( O_{t,i} \).

By choosing \( \alpha \gg 1 - \alpha \) we achieve that the similarity between two behaviors is primarily rated based on their shared dimensions. In the best case \( C_{t+1,j} \) retains all dimensions and covers many additional ones (distance of 0). In the worst case we have nearly no shared dimensions and no additional ones (distance of 1.0).

**Similarity based on statistical characteristics.** Besides the subspace similarity, the actual values within these dimensions are important. For ex-
ample, solely because two clusters share a dimension like ‘temperature’, their values can differ extremely (high vs. low temperature); these behaviors should not be mapped. A small change in the values, however, is possible for evolving behaviors. Considering a spatial dimension, this change corresponds to a slight cluster movement.

Given a cluster $C = (O, S)$, we denote the set of values in dimension $d$ with $v(C, d) = \{ o[d] \mid o \in O \}$. The similarity between two clusters $C_{t,i} = (O_{t,i}, S_{t,i})$ and $C_{t+1,j} = (O_{t+1,j}, S_{t+1,j})$ is thus achieved by analyzing the corresponding sets $v(C_{t,i}, d)$ and $v(C_{t+1,j}, d)$. In many applications, normal distributions are well suited to model the values a cluster follows; this is often exploited, for example by clustering based on maximizing the data’s likelihood assuming a mixture of normal distributions [DLR77]. Thus, without losing much information, we can represent the sets $v(C_{t,i}, d)$ and $v(C_{t+1,j}, d)$ by two normal distributions $X_d$ and $Y_d$ with means $\mu_x, \mu_y$ and variances $\sigma_x, \sigma_y$. The similarity can now be measured by comparing these distributions. We use the information theoretic Kullback-Leibler divergence ($KL$). Informally, we calculate the expected number of bits required to encode a new distribution of values at time step $t+1$ ($Y_d$) given the original distribution of the values at time step $t$ ($X_d$). By using our cluster approximations $\mu_x, \mu_y, \sigma_x, \sigma_y$, we can calculate the KL via a closed-form expression:

$$KL(Y_d || X_d) = \ln \left( \frac{\sigma_x}{\sigma_y} \right) + \frac{\sigma_y^2 + (\mu_y - \mu_x)^2}{2\sigma_x^2} - \frac{1}{2} =: KL(C_{t,i}, C_{t+1,j}, d)$$

The $KL$ is an asymmetric measure, i.e. $KL(Y_d || X_d) \neq KL(X_d || Y_d)$; therefore it is suitable for a temporal tracing model, in which time progresses in one direction. By using the $KL$, we do not just account for the absolute deviation of the means, but we have also the advantage of including the variances. A behavior with a high variance in a single dimension allows a higher evolution of the means for successive similar behaviors. A small variance of the values, however, only permits a smaller deviation of the means. This is illustrated in Figure 5.3.

We use the $KL$ to calculate the similarity per dimension, and the overall similarity is attained by cumulating over several ones. Apparently, we just
5.3. Model for Subspace Cluster Tracing

Figure 5.3: Exemplary contour lines for the \( KL \) distance between normal distributions \( A, B \) with constant \( \mu_A - \mu_B \); the corresponding variances are plotted on the \( x \)- and \( y \)-axis; it holds \( a > b > c > d > e \).

have to use dimensions that are in the intersection of both clusters. The remaining dimensions are non-relevant for at least one cluster and hence are already penalized by our subspace distance function. Our first approach for computing the similarity based on statistical characteristics is

\[
V(C_{t,i}, C_{t+1,j}, I) = \frac{\sum_{d \in I} KL(C_{t,i}, C_{t+1,j}, d)}{|I|} \tag{5.1}
\]

with \( I = S_{t,i} \cap S_{t+1,j} \) for averaging.

In a perfect scenario this distance is a good way to trace behaviors. In practice, however, we face the following problem: Consider the example in Figure 5.4 (note the 7-dimensional space). With our clustering we identify the cluster \( C_{1,2} \) at time step \( t = 1 \) and the cluster \( C_{2,2} \) with the same relevant dimensions in \( t = 2 \). However, \( C_{2,2} \) is shifted in dimensions \( d_1 \) and \( d_2 \); the distance function proposed above (Eq. 5.1) would determine a very high value and hence the behaviors would not be mapped. A large part \( \{d_3, ..., d_7\} \) of the shared relevant dimensions \( \{d_1, ..., d_7\} \), however, show nearly the same characteristics in both clusters. The core of the behaviors is completely identical, and therefore a mapping is reasonable; this is illustrated by the mapping of \( C_{1,2} \) to \( C_{2,2} \) in the lower part of Figure 5.4. Consider another example: The core of two clusters detected in the oceans is identical, for example, their biomass and temperature are similar. However, the clusters are located on different hemisphere so that their ocean currents are different. These additional, non-core, dimensions provide us with further information about the single clusters at their current time step. They are mainly induced by the
individual populations and are obtained by subspace clustering. For the continuation of the behavior, however, these dimensions are not important. Note that non-core dimensions are a different concept than non-relevant ones; non-core dimensions are shared relevant ones with differing values.

An effective distance function between clusters identifies the core of the behaviors and incorporate it into the distance. We accomplish this by using a subset $Core \subseteq S_{t,i} \cap S_{t+1,j}$ for comparing the values in Equation 5.1 instead of the whole intersection $S_{t,i} \cap S_{t+1,j}$. Unfortunately, this subset is not known in advance, and it is not reasonable to choose a fixed threshold that excludes some dimensions from the distance calculation if the corresponding dissimilarity is too large. Thus, we develop a variant that automatically determines the core. The idea is to choose the ‘best’ core among all possible cores for the given two clusters. That is, for each possible core we determine the distance with respect to their value distributions, and we additionally penalize dimensions that are not included in the core. The core with the smallest overall distance is selected; that is, we trade off the size of the core against the value $V(C_{t,i}, C_{t+1,j}, Core)$:

Figure 5.4: Example for core dimension concept in a 7-dimensional space.
Definition 5.5 **Core-Based Distance Function for Values.** The core-based distance function with respect to values for two clusters $C_{t,i} = (O_{t,i}, S_{t,i})$ and $C_{t+1,j} = (O_{t+1,j}, S_{t+1,j})$ is defined by

$$V(C_{t,i}, C_{t+1,j}) = \min_{Core \subseteq S_{t,i} \cap S_{t+1,j}} \left\{ \beta \cdot \left| \text{NonCore} \right| \left| S_{t,i} \cap S_{t+1,j} \right| + (1 - \beta) \cdot V(C_{t,i}, C_{t+1,j}, Core) \right\}$$

with the penalty factor $\beta \in [0, 1]$ for the non-core dimensions $\text{NonCore} = (S_{t,i} \cap S_{t+1,j}) \setminus \text{Core}$.

By selecting a smaller core, the first part of the distance formula gets larger. The second part, however, gets the possibility of determining a smaller value. The core must comprise at least one dimension; otherwise, we could map two clusters even if they have no dimensions with similar characteristics.

**Overall distance function.** To correctly identify the evolving clusters we have to consider evolutions in the relevant dimensions as well as in the value distributions. Accordingly, we use both distance measures simultaneously. Again, we require that two potentially mapped clusters share at least one dimension; otherwise, these clusters cannot represent similar behaviors.

Definition 5.6 **Overall Distance Function.** The distance function comprising subspace and core-based value similarity, for two clusters $C_{t,i} = (O_{t,i}, S_{t,i})$ and $C_{t+1,j} = (O_{t+1,j}, S_{t+1,j})$ with $|S_{t,i} \cap S_{t+1,j}| > 0$, is defined by

$$\text{dist}(C_{t,i}, C_{t+1,j}) = \gamma \cdot V(C_{t,i}, C_{t+1,j}) + (1 - \gamma) \cdot S(C_{t,i}, C_{t+1,j})$$

with $\gamma \in [0, 1]$. In the case of $|S_{t,i} \cap S_{t+1,j}| = 0$, the distance is set to $\infty$.

5.3.3 Clustering for Improved Tracing

In the previous sections we assume a given clustering per time step so that we can determine the distances and the mapping graph. In general, our tracing model is independent of the underlying clustering method; it can flexibly be chosen based on the current application. However, since there are temporal relations between consecutive time steps, we develop a clustering method...
whose accuracy is improved by exploiting these relations and that avoids unstable clusterings (i.e., highly varying clusterings in consecutive time steps). In our subspace clustering definition, we adopt the cell-based clustering paradigm [PJAM02, YM03], because approaches from this paradigm show high quality results and are efficiently computable [MGAS09]. A benefit of cell-based algorithms is that the representing cells can be positioned flexible in space, i.e. they do not need a fixed grid as other approaches. Basically, we approximate clusters by hypercubes in the data space. The extent of a hypercube is restricted to \( w \) in the relevant dimensions of a cluster and unrestricted in the non-relevant ones. Thus, the values of the objects vary to at most \( w \) within the relevant dimensions and hence they represent a meaningful grouping. Additionally, we require that a valid cluster summarizes at least \( minSup \) many objects.

**Definition 5.7 Hypercube and Valid Subspace Cluster.** A hypercube \( H_S \) with the relevant dimensions \( S \) is defined by lower and upper bounds

\[
H_S = [low_1, up_1] \times [low_2, up_2] \times \ldots \times [low_D, up_D]
\]

with \( up_i - low_i \leq w \) \( \forall i \in S \) and \( low_i = -\infty, up_i = \infty \) \( \forall i \notin S \). The mean of \( H_S \) is denoted by \( m_{H_S} \). The hypercube \( H_S \) represents all objects \( \text{Obj}(H_S) \subseteq DB \) with \( o \in \text{Obj}(H_S) \iff \forall d \in \{1, \ldots, D\} : low_d \leq o[d] \leq up_d \). A subspace cluster \( C = (O, S) \) is valid iff there exists a hypercube \( H_S \) with \( \text{Obj}(H_S) = O \) and \( |\text{Obj}(H_S)| \geq minSup \).

In the next paragraphs we introduce how temporal relations between time steps can be exploited to improve the tracing accuracy and to avoid unstable clusterings.

**Predecessor information.** We assume that the initial clustering at time step \( t = 1 \) is known. (We discuss this later.) Caused by the temporal aspect of our data, clusters at a time step \( t \) occur with high probability also in the next time step — not identical, but similar. Consequently, given a cluster and the corresponding hypercube \( H_S \) at time step \( t \), we try to find a cluster at the subsequent time step in a similar region. This is achieved by a Monte Carlo
approach: we draw a random point \( m_{t+1} \in \mathbb{R}^D \) that represents the initiator of a new hypercube and that is located nearly to the mean \( m_{H_S} \) of \( H_S \).

**Definition 5.8 Initiator of a Hypercube.** A point \( p \in \mathbb{R}^D \), called initiator, together with a width \( w \) and a subspace \( S \) induces a hypercube \( H^w_S(p) \) defined by \( \forall d \in S : \text{low}_d = p[d] - \frac{w}{2}, \text{up}_d = p[d] + \frac{w}{2} \) and \( \forall i \notin S : \text{low}_i = -\infty, \text{up}_i = \infty \).

Accordingly, by using this initiator concept, we apply the cell-based clustering paradigm. After inducing a hypercube by an initiator, we check if the corresponding cluster is valid. Formally, the initiator \( m_{t+1} \) is drawn from the region \( H^w_S(m_{H_S}) \); that is, we permit a change of the cluster. The new hypercube is then induced by \( m_{t+1} \), i.e. the generated cluster is \( H^w_S(m_{t+1}) \).

With this method we detect changes in the values; however, also the relevant dimensions of a cluster can change: The initiator \( m_{t+1} \) can induce different hypercubes for different relevant dimensions \( S \). For example, all or just one dimension of the hypercube could be restricted to the maximal extent \( w \). Therefore, beside the initiator \( m_{t+1} \), we additionally have to determine the relevant subspace of the new cluster. We discuss both in the following.

**Determining the best cluster.** A first possible approach is to use a quality function \([PJAM02, YM03, GKS10]\): \( \mu(H_S) = \text{Obj}(H_S) \cdot k_{|S|} \). The more objects or the more relevant dimensions are covered by the cluster, the higher is its quality. These objectives are in conflict; therefore a trade-off is realized with the constant parameter \( k \). In time step \( t + 1 \) we could simply choose the subspace \( S \) such that the hypercube \( H^w_S(m_{t+1}) \) maximizes \( \mu(H^w_S(m_{t+1})) \).

This method, however, optimizes the quality of each single cluster; it is not intended to find good tracings. Possibly, the distance between each cluster from the previous clustering \( Clus_t \) and our new cluster is large, and we would find no similar behaviors. Thus, we directly integrate the distance function \( \text{dist} \) into the quality function, i.e. we want to prefer clusters leading also to small mapping distances. Consequently, we choose the subspace \( S \) such that the hypercube \( H^w_S(m_{t+1}) \) maximizes our novel distance-based quality function.
Definition 5.9 Distance-Based Quality Function. Given the hypercube $H_S$ in subspace $S$ and a clustering $Clus_t$, the distance based quality function is

$$q(H_S) = \mu(H_S) \cdot (1 - \min_{C_t \in Clus_t} \{\text{dist}(C_t, C_S)\})$$

where $C_S$ indicates the induced subspace cluster of the hypercube $H_S$.

We enhance the quality of the clustering by selecting a set of possible initiators $M$ from the specified region; this is also important as the direction of a cluster change is not known in advance. From the resulting set of potential clusters, we select the one that has the highest quality.

Overall we realize that for each cluster $C \in Clus_t$ a potential temporal continuation is identified in time step $t + 1$. Nonetheless it is also possible that our method identifies no valid hypercube for a single cluster $C \in Clus_t$, e.g. because too few objects are located around the selected initiator. This indicates that a behavior type has disappeared in the current time step.

Uncovered objects and the initial clustering. If a behavior disappears or emerges, there will be some objects of the current time step that are not part of any identified cluster. In other words: if we denote the set of clusters generated so far by $Clus_{t+1}$, the set $\text{Remain}_{t+1} := DB_{t+1} \setminus \bigcup_{C_t \in Clus_{t+1}} O_t$ can still contain objects and thus clusters. Especially for the initial clustering at time step $t = 1$ we have no predecessor information and hence $Clus_1 = \emptyset$ at the start. To discover as many patterns as possible, we have to check if the objects within $\text{Remain}_{t+1}$ induce novel clusters. We cannot infer the initiators of possible hypercubes based on previous clusters; instead, we use the remaining objects itself as initiators for the hypercubes. We draw a set of initiators $M \subseteq \text{Remain}_{t+1}$, where each $m \in M$ induces a set of hypercubes $H^S_w(m)$ in different subspaces. Finally, we choose the hypercube that maximizes our quality function. If this hypercube corresponds to a valid cluster, we add it to $Clus_{t+1}$, and thus the set $\text{Remain}_{t+1}$ is reduced. This procedure is repeated until no valid cluster is identified or the set $\text{Remain}_{t+1}$ is empty. Note that our method has the advantage of generating overlapping clusters. We select the initiators from the set $\text{Remain}_{t+1}$; the objects covered by the
5.3. Model for Subspace Cluster Tracing

Hypercubes, however, are a subset of the whole database. Thereby we realize a meaningful non-partitioning clustering.

**Algorithm 5.1:** Processing scheme of subspace cluster tracing method

```plaintext
method: main(databases $DB_1, \ldots, DB_T$)
1 $G = (V, E, w)$ // mapping graph, empty at beginning
2 $Clus_0 = \emptyset, \ldots, Clus_T = \emptyset$ // no clusters determined
3 for $t = 1, \ldots, T$ do
4 $Remain = DB_t$ // all objects unclustered
5 for $C = (O, S) \in Clus_{t-1}$ do // predecessor information
6 determine hypercube $H_S$ of $C$ and its mean $m_{H_S}$
7 randomly draw a set $M$ of initiators with $m \in M$ ⇒ $m \in H_S(w(m_{H_S}))$
8 $C^* \leftarrow \text{ClusterAndMappingDetection}(M, t)$
9 if $C^* \neq \bot$ then // let $C^* = (O^*, S^*)$
10 $Clus_t = Clus_t \cup \{C^*\}$
11 $Remain = Remain \setminus O^*$
12 while $Remain \neq \emptyset$ // still unclustered objects
13 randomly draw initiators $M \subseteq Remain$
14 $C^* \leftarrow \text{ClusterAndMappingDetection}(M, t)$
15 if $C^* \neq \bot$ then // let $C^* = (O^*, S^*)$
16 $Clus_t = Clus_t \cup \{C^*\}$
17 $Remain = Remain \setminus O^*$
18 else // all clusters detected, next time step
19 break;
20 return $G$;
```

method: ClusterAndMappingDetection(initiators $M$, time step $t$)
21 // —— calculate best cluster ——
22 $(m^*, S^*) = \arg \max_{(m, S) \in M \times P \{1, \ldots, D\}}\{q(H_S^w(m)) \text{ if } |Obj(H_S^w(m))| \geq \min\text{Sup} \text{ else } -1\}$
23 if $|Obj(H_S^w(m^*))| < \min\text{Sup}$ then return $\bot$ // only invalid clusters
24 $C^* = (Obj(H_S^w(m^*)), S^*)$ // the novel cluster
25 // —— update mapping graph ——
26 $V = V \cup \{C^*\}$ // new node in mapping graph
27 for $C_{t-1} \in Clus_{t-1}$ do
28 if $\text{dist}(C_{t-1}, C^*) < \tau$ then
29 $E = E \cup \{(C_{t-1}, C^*)\}$ // new edge in mapping graph
30 return $C^*$;
```

**Summary.** The overall processing scheme of our algorithm is illustrated in Algorithm 5.1. For each point in time (line 3) we first perform clustering based on the given predecessor information (lines 5-11), followed by our method to detect emerging clusters (lines 12-19). The actual clusters and corresponding mappings between clusters are detected in lines 21-30, using
our distance-based quality function. Overall, our clustering method specifically utilizes the advantages of temporal data to obtain high quality temporal continuations by nesting mapping and clustering: We steer the cluster identification to regions in the data space where good clusters are expected. Thus, cluster tracing is no longer independent of the clustering but we integrate model specific properties in this step.

### 5.3.4 Computational Aspects

In the following we briefly analyze the computational complexity of our model. Essentially, we can distinguish two phases within the method: the actual clustering of each time step and the calculation of mapping distances. Since in general our model is independent of the underlying clustering algorithm and thus any choice would be possible, we focus on the second aspect. Anyhow we want to mention the exponential complexity of many subspace clustering algorithms with respect to the dimensionality of clusters hidden in the data. Thus, this clustering step is usually the determining factor for the runtimes in practical applications of the method.

For determining a mapping graph consisting of $T$ time steps, we have to calculate the mapping distances between $T - 1$ many pairs of clusterings. For two successive clusterings $Clus_t$ and $Clus_{t+1}$ we have to determine $|Clus_t| \cdot |Clus_{t+1}|$ distance values according to Definition 5.6 in which the value based distance function is the more complex summand, because the optimal core is identified by a minimization procedure (cf. Definition 5.5). If the intersection between the relevant dimensions of two clusters has a cardinality of $i$, $2^i - 1$ potential cores have to be checked. Let $k$ and $l$ be the average dimensionality for the clusters at time step $t$ and $t+1$ respectively. By assuming that the relevant dimensions are randomly drawn from all dimensions $\{1, \ldots, D\}$, the number of cluster pairs whose intersection has a cardinality of $i$ can be determined by binomial coefficients. By averaging over all possible pairs, we get the expected number of cores to be tested given two clusters with dimensionality $k$ and $l$:
Summarized, the overall number of cores to be tested and thus the overall number of value based similarity calculations for all time steps is given by

\[ T - 1 \sum_{t=1}^{T-1} |Clus_t| \cdot |Clus_{t+1}| \cdot O(2^{\min\{a_t, a_{t+1}\}}) \leq \left( T - 1 \right) \cdot \max_{t \in \{1, \ldots, T\}} \left\{ |Clus_t|^2 \right\} \cdot O(2^{a_{\text{max}}}) \]

with average dimensionality \( a_t := \frac{\sum_{C_t,i \in Clus_t} \frac{|S_{t,i}|}{|Clus_t|}} \) and maximal average dimensionality \( a_{\text{max}} := \max_{t=1, \ldots, T} \{a_t\} \). Thus, our method scales linear with the number of points in time but exponential with the cluster dimensionality.

![Figure 5.5: Scalability with respect to the cluster dimensionality.](image)

As a proof of concept, we generate synthetic data comprising two time steps each with 1,000 objects and 10 hidden clusters. In Figure 5.5 we depict the runtime of our method for an increasing number of relevant dimensions per cluster. While the runtime of our approach increases exponentially with respect to the number of relevant dimensions, the absolute runtime is still acceptable. Moreover, since in real world data the cluster dimensionality is often much smaller than the data dimensionality, we believe that our method is applicable to a broad range of datasets.
5.4 Experiments

To evaluate the tracing quality we use real world and synthetic data. For real world data we use scientific grid data reflecting oceanographic characteristics as temperature and salinity of the oceans. It contains 20 time steps, 8 dimensions, and 71,430 objects. The synthetic data covers 24 time steps and 20 dimensions. In average, each time step contains 10 clusters with 5-15 relevant dimensions. Since we hide all kinds of developments (emerge, merge, split-up, or disappear) and evolution (subspace and value changes) within this data, these values are slightly changed. In our experiments we focus on the quality of our approach. In synthetic datasets the correct mappings between the clusters are given. Based on the detected mappings of our approach we calculate the precision and recall values: we check whether our approach detects all but only the true mappings between clusters. For tracing quality we use the F1 value corresponding to the harmonic mean of recall and precision. Our approach tackles the problem of tracing clusters with varying subspaces and is based on object-value-similarity. Even if we would constrain our approach to handle only full-space clusters as existing solutions, such a comparison is only possible when we artificially add object ids to the data (to be used by these solutions). Tracing clusters based on these artificial object ids, however, cannot reflect the ground truth in the data. Summarized, comparisons to other approaches are not performed since it would be unfair. We use Opteron 2.3GHz CPUs and Java6 64bit.

5.4.1 Tracing quality

In this section we analyze how the different parameters of our algorithm affect the cluster tracing effectiveness. The influence of $\gamma$ is evaluated in Figure 5.6 for three different values of $\tau$ using synthetic data. By $\gamma$ we control the trade-off between subspace similarity and value similarity in our overall distance function (cf. Definition 5.6 on page 93). The objective of this function is to allow that clusters can gain or lose dimensions, and also to allow

---

$\dagger$ provided by the Alfred Wegener Institute for Polar and Marine Research, Germany
5.4. Experiments

Figure 5.6: Tracing quality for different \( \gamma \) and \( \tau \).

Figure 5.7: Evaluation of core dimension concept.

Figure 5.8: Influence of varying cluster center shift.

Figure 5.9: Influence of input clustering quality.

that cluster object values can slightly shift. Obviously we want to prevent extreme cases for a meaningful tracing, i.e., subspace similarity with no attribute similarity at all (\( \gamma \to 0 \)), or the other way round. This is confirmed by the figure, as the tracing quality highly degrades, if \( \gamma \) reaches 0 or 1 for all \( \tau \) values. As \( \gamma = 0.3 \) enables a good tracing quality for all three \( \tau \), we use this as a default value. Note that with the threshold \( \tau \) we can directly influence how many cluster mappings are created. The figure shows that \( \tau = 0.1 \) is a good trade-off and is thus used as default value. With a bigger \( \tau \) the tracing quality worsens: too many mappings are created and we cannot distinguish between meaningful or meaningless mappings. The same is true for \( \tau \to 0 \): no clusters are mapped and therefore the clustering quality reaches zero; thus we excluded plots for \( \tau \to 0 \).

The concept of core dimensions is evaluated in Figure 5.7. We analyze the influence on the tracing quality (left axis) with a varying \( \beta \) on the x-axis; i.e., we change the penalty for non-core dimensions. The non-core
dimensions are those shared dimensions of two compared clusters in which large changes are allowed. Remember, non-core dimensions are a different concept than non-relevant ones; non-core dimensions are shared relevant dimensions with differing values. The higher the penalty, the more dimensions are included in the dimension core; i.e., more shared dimensions are used for the value-based similarity. In a second curve, we show the absolute number of non-core dimensions (right axis) for the different penalties: the number decreases with higher penalties. In this experiment the exact number of non-core dimensions in the synthetic data is 10. We can draw the following conclusions regarding tracing quality: A forced usage of a full core (all shared dimensions, $\beta \rightarrow 1$) is a bad choice, as there can be some shared dimensions with different values. By lowering the penalty we allow some dimensions to be excluded from the core and thus we can increase the tracing quality. With $\beta = 0.1$ the highest tracing quality is obtained; this is plausible as the number of non-core dimensions corresponds to the number that is existent in the data. A too low penalty, however, results in excluding nearly all dimensions from the core (many non-core dimensions, $\beta \rightarrow 0$) and hence the quality drops for this case. In the experiments, we use $\beta = 0.1$ as a default value.

The objective of our tracing approach is to map clusters of similar behavior, i.e. two clusters are mapped if the corresponding object values are similar; thus, strong value differences in core dimensions of two compared clusters should prevent a mapping. This is evaluated in Figure 5.8 with synthetic data. On the x-axis, the average cluster center shift between consecutive time steps is plotted. The figure shows that with greater shifts less clusters are mapped and thus the tracing quality degrades. It can also be seen that this effect can be counterbalanced with a higher value of $\tau$; by this, greater cluster center shifts are allowed.

The effect of input clustering quality on cluster tracing quality is evaluated in Figure 5.9. We analyze how mappings between clusters are affected in the case that hidden clusters are incorrectly identified by the clustering algorithm. This is achieved by a varying hypercube width in our clustering model. The figure shows that clustering quality (measured via RNIA
5.4. Experiments

Figure 5.10: Comparison with and without predecessor information & distance quality function.

Figure 5.11: Cumulated number of evolutions & developments over 24 time steps on synthetic data.

[PM06]) and tracing quality are highly correlated; a decreasing clustering error results in an increasing tracing quality. Thus, for meaningful cluster tracing a clustering algorithm providing high-quality clusters is essential.

5.4.2 Detection of behavior developments

In the following, we analyze whether our model is able to detect the different behavior developments. Up to now, we used our enhanced clustering method that utilizes the predecessor information and the distance based quality function. Now, we additionally compare this method with a variant that performs clustering of each step independently. Intuitively, this corresponds to the idea that each time step is the first one, i.e. without preceding information. In Figure 5.10 we use the oceanographic dataset and we determine for each time step the number of disappeared behaviors for each clustering method. The experiment indicates, that the number of unmapped clusters for the approach without any predecessor or distance information is larger than for our enhanced approach. By transferring the clustering information between the time steps, can map a larger amount of clusters from one time step to the next. We map clusters over a longer time period; thus, yielding a more meaningful tracing of evolving clusters.

The aim of tracing is not just to map similar clusters but also to identify different kinds of evolution and development. In Figure 5.11 we plot the number of clusters that gain or lose dimensions and the four kinds of de-
Figure 5.12: Number of evolutions and developments on real world data; left: cumulated over 20 time steps, right: for each time step.

development cumulated over all time steps. Beside the detected numbers, we show the intended number based on this synthetic data. The first four bars indicate that our approach is able to handle dimension gains or losses; i.e., we enable subspace cluster tracing, which is not considered by other models. The remaining bars show that also developments can be accurately detected. Overall, the intended transitions are found by our tracing. In Figure 5.12 we perform a similar experiment on real world data. We report only the detected number of patterns because exact values are not given. On the left we cumulate over all time steps. Again, our approach traces clusters with varying dimensions. Accordingly, on real world data it is a relevant scenario that subspace clusters lose some of their characteristics. Thus, it is mandatory to use a tracing model that handle these cases, as our model does. The developments are also identified in this real world data. To show that the effectiveness is not restricted to single time steps, we analyze the detected patterns for each time step individually on the right. Based on the almost constant slopes of all curves, we see that our approach performs effectively.

5.5 Application Scenario

To demonstrate that our tracing approach detects reasonable mappings on real world data, a tracing result for the oceanographic grid data is shown in Figure 5.13. In the figure, different colors correspond to different clusters. Our method detects and traces several of the oceanic provinces [Lon98]. For example, a cluster similar to the Pacific Equatorial Divergence Province is
5.6 Conclusion

In this chapter, we proposed a model for tracing evolving subspace clusters in high dimensional temporal data. In contrast to existing methods, we trace clusters based on their characteristics; that is, clusters are not mapped based on the fraction of objects they have in common, but on the similarity of their corresponding object values. A good application example is oceano-

Figure 5.13: Clusterings of oceanographic grid data for January and February and one mapping.

found in both time steps and our method accomplishes a mapping between these clusters; we illustrate this mapping in the figure.

Another traceable cluster is a cluster representing the ocean’s low productivity regions located at the position of the subtropical gyres. We analyzed the temporal behavior of this cluster over a period of one year, and four months are illustrated in Figure 5.14 (left). A selection of the corresponding values per time step is given on the right. Noticeable in this figure is the increase of chlorophyll, net primary production, and silicate in September. This could indicate a connection to phytoplankton blooms [SDY02] occurring regularly in temperate and sub-polar water areas. In winter, waters are well mixed and nutrients circulate up from bottom waters. As soon as the ocean warms in late spring, the warm water will stay at the top of the water column as it is less dense. At the same time light level increases and phytoplankton population grows exponentially. In most cases the available nutrients are used up within weeks or months. Sometimes a second bloom occurs at autumn.
Figure 5.14: Tracing of a single cluster over a one year period; left: illustrations of Jan., Apr., Jul., and Oct.; right: selection of the corresponding values per time step.

graphic grid data, where values are recorded by sensors, which are positioned according to a fixed raster. For maximal flexibility, our tracing approach distinguishes several developments of behavior. We enable effective tracing by introducing a novel distance measure that determines the similarity between clusters; this measure comprises subspace and value similarity, reflecting how much a cluster has evolved. In the experimental evaluation we showed that high quality tracings are generated.

This finishes the second part of this thesis, in which we developed techniques for finding relationships between given clusterings. The effectiveness of these techniques depends on the applied mapping components that map clusters in one cluster set to the most-fitting counterparts in a second cluster set. Applications are the evaluation of stream clustering results and the tracing of evolving subspace clusters over time.

In Part I and II, we introduced mining techniques for temporal data. The focus was on the quality of generated patterns and the quality of found mappings between patterns. The core of mining tasks like clustering are similarity estimates between analyzed objects. In the area of time series similarity search, effective similarity models have been proposed. Due to the vast amounts of similarity queries in data mining applications, and the importance for rapid feedback in similarity search systems, the efficiency of similarity search queries is highly important. In the next part, we give an introduction to similarity search and efficient processing techniques.
Part III

Introduction to Similarity Search
Chapter 6

Distance-Based Similarity Search: Introduction

In this part, we will give a general overview over similarity search, and the following Part IV will cover the specific topic of time series similarity search.

Similarity search is an active research area with a wide range of application domains. It is typically realized in a content-based fashion, in which similarity between objects is measured by distance functions. In both similarity-based data mining and stand-alone similarity search, rapid processing of similarity queries is crucial. We concentrate on this aspect in the remaining parts of the thesis.

In this chapter, we give a brief introduction to distance functions and types of similarity queries, and we present established concepts for efficient processing of such queries, namely multistep filter-and-refine processing and indexing structures. In the second and final chapter of this part, we develop an indexing structure for complex data that serves as an example for both concepts and how they can be tightly integrated.

6.1 Distance Functions and Similarity Queries

In distance-based similarity search, the objects are represented by content-related features, as for example color histograms and time series, and sim-
ilarity between objects is modeled by distance functions operating in the corresponding feature space.

**Definition 6.1** Given an object set $F$, a **distance function** $dist$ is a mapping

$$F \times F \rightarrow \mathbb{R}^+_0$$

reflecting object (dis)similarity by assigning object pairs to a non-negative value. The larger the distance, the more dissimilar are the objects. For an effective notion of similarity, the following properties are highly desired $\forall p, q, s \in F$:

- **definiteness**: $dist(q,p) = 0 \iff q = p$
- **symmetry**: $dist(q,p) = dist(p,q)$
- **triangle inequality**: $dist(q,p) \leq dist(q,s) + dist(s,p)$

If all three properties are fulfilled, a distance measure is called a metric.

In the following, we will assume objects from a multidimensional vector space for the set $F$, and for a database $DB$ and a query object $q$ it holds that $\{q\} \cup DB \subseteq F$. An illustration of a 4-dimensional vector space with two points and a distance function is illustrated on the right.

The metric properties are not only important for an effective notion of similarity, i.e. a correspondence to human perception, but they are also important with respect to efficient query processing; for example, metric indexing structures rely on the triangle inequality (cf. Section 6.2.2). A widely-used metric distance function is the Minkowski distance, also known as $L_p$-Norm distance.

**Definition 6.2** Given two objects $s = (s_1, \ldots, s_d)$ and $t = (t_1, \ldots, t_d)$ from a $d$-dimensional vector space, the **Minkowski Distance** is defined as

$$dist_{L_p}(s,t) = \left( \sum_{i=1}^{d} |s_i - t_i|^p \right)^{1/p}$$
6.1. Distance Functions and Similarity Queries

For \( p = 2 \) it corresponds to the Euclidean distance. The Euclidean distance is more sensitive to noise than the Manhattan distance \((p = 1)\), which does not emphasize dissimilarities as much. The Chebyshev distance is obtained for \( p = \infty \), with \( \text{dist}_{L_{\infty}} = \max_{1 \leq i \leq d} (|s_i - t_i|) \). It emphasizes the most dissimilar features.

In distance-based similarity search, the task is to find similar objects to a given query object \( q \), with respect to a database \( DB \) and a distance function \( \text{dist} \). There are several different definitions of similarity queries, and in this thesis, we use range and \( k \)-nearest-neighbor queries.

For a range query, the user has to define a specific distance threshold, and the result set contains all database objects whose distance to the query object is below or equal to that threshold.

**Definition 6.3** Given a query object \( q \), an object database \( DB \), a distance function \( \text{dist} \), and a distance range \( \epsilon \), a **Range Query** is defined by

\[
\text{Range}_\epsilon(q) = \{ o \in DB \mid \text{dist}(q, o) \leq \epsilon \}
\]

While range queries are suitable in some domains, choosing an appropriate threshold can be tedious for the user. In these scenarios, a \( k \)-nearest-neighbor (\( k \)NN) query is more suitable, which calculates the \( k \) most similar objects to the query object.

**Definition 6.4** Given a query object \( q \), an object database \( DB \), a distance function \( \text{dist} \), and a result size \( k \), a **k-Nearest-Neigbor Query** is defined as

\[
\text{NN}_k(q) = \{ R \subseteq DB \mid \forall o \in R, \forall o' \in DB \setminus R : \text{dist}(q, o) \leq \text{dist}(q, o') , | R | = k \}
\]

Technically, a \( k \)NN query result can be determined by incrementally updating an intermediate result set \( R' \) with \( |R'| \leq k \). If an object \( o \in DB \) with \( \text{dist}(q, o) < \max_{p \in R'} \{ \text{dist}(q, p) \} \) is found during query processing, \( o \) replaces the object in \( R' \) with the largest distance to the query \( q \). If it can be ruled out that another object with a lower distance exists (e.g., the whole \( DB \) was processed), query processing is finished and \( R = R' \).
In the next section, we discuss existing work on efficient processing of similarity queries.

6.2 Efficient Query Processing

Similarity queries can be processed in linear time by sequentially scanning the database. In many applications, however, fast query times are crucial and long waiting periods are prohibited. Besides a sequential scan, reasons for long runtimes of similarity queries can be distance functions of high computational complexity, high dimensionalities of data items, or the sheer size of an analyzed database.

The most prominent concepts in database research for efficient similarity search query processing are indexing structures and multistep filter-and-refine frameworks. Their objective is to minimize the number of distance computations to objects not belonging to a query’s final result set, corresponding to substantial reduction of query times. We give a brief overview over both concepts, and when required, further details will be given in the corresponding chapters.

6.2.1 Multistep Filter-and-Refine Processing

Multistep filter-and-refine frameworks [AFS93, KSF*96, SK98] prevent unnecessary distance computations by rejecting database objects based on filter functions, which are approximations of the actual distance functions. A filter function is used to quickly retrieve a candidate set of objects from the database, corresponding to the filter step; in the refinement step, the actual distances to the query object are computed to obtain the final result.

Rejection of objects is based on a specific pruning threshold. It is either the distance range $\epsilon$ of a range query, or in case of a $k$NN query, it is the largest distance from the query to the current nearest neighbors in the intermediate result set. The basic idea is illustrated in Figure 6.1, where we assume a pruning threshold of $\epsilon$. The applied distance function and the corresponding filter function are denoted as $\text{dist}$ and $\text{dist}_{\text{filter}}$. In both steps, i.e.
filter and refinement, many database objects are pruned.

To ensure that query processing via multistep filter-and-refine algorithms yields the desired efficiency gains, a number of quality criteria should be fulfilled; in [AWS06b], these criteria were termed ICES: Indexability, Completeness, Efficiency, and Selectivity. A filter should be indexable, i.e., it should be applicable as a distance function in an indexing structure. Completeness of the filter step ensures that efficiency is not at the cost of correctness of the result, implying that there are no false dismissals; in multistep filter-and-refine frameworks, this is ensured by the lower-bounding property of the applied filter function [Fal96, SK98].

**Definition 6.5** Given a distance function \( \text{dist} \), a function \( \text{dist}_{\text{filter}} : F \times F \to \mathbb{R}_0^+ \) is a **Lower Bound** to \( \text{dist} \) iff \( \forall q, p \in F : \text{dist}_{\text{filter}}(q, p) \leq \text{dist}(q, p) \).

Accordingly, if the distance with respect to a lower bound exceeds the pruning threshold, then the exact distance does so as well. Hence, lossless pruning is possible. If the filter distance is below the pruning threshold, the exact distance is computed. A filter function should be efficiently computable (i.e., its runtime should be significantly smaller than the runtime of the exact distance computation). Finally, a good filter function should be selective (i.e., the number of generated candidates should be small).

Filter functions can be obtained by reducing either the computational complexity of the distance function or the dimensionality of the data. Computational complexity of distance functions can be reduced, for example, by
relaxing the constraints of an optimization problem [AWS06a]. Reducing
the data dimensionality corresponds to fewer computations in each distance
calculation. A prominent example for dimensionality reduction is Principal
Component Analysis [Jol02]. It exploits correlations between dimensions
corresponding to redundancy in the data and transforms the original space
to a lower-dimensional one containing most of the relevant information.

Several filter-and-refine frameworks exist in the literature. Range queries
are handled in [AFS93]. k-NN queries are processed in [KSF+96] and [SK98].
The approach in [SK98] is optimal with respect to the number refinements,
and it is also the foundation of the query processing algorithm in following
chapter’s indexing structure. In [HMNO12], an approximate filter-and-refine
algorithm for k-NN queries was presented that greatly reduces the number of
candidates and that is parameterizable by the success rate of correct answers.
Novel filter-and-refine techniques are developed in Chapters 7, 9, and 10 of
this thesis. Filter-and-refine frameworks are usually combined with indexing
structures.

6.2.2 Indexing Structures

In this section we give a quick introduction to the concept of indexing struc-
tures. An overview and in-depth information can be found in [Sam05]. In-
dexing structures have the same objective as filter-and-refine frameworks: to
reduce the number of distance computations to database objects. The ba-
sic strategy, however, is different. Filter-and-refine prunes individual objects
from the list of possible candidates, based on an object’s filter distance to
the query. Indexing structures, in contrast, prune several objects at the same
time using compact information of these objects. Exceptions are structures
such as the VA-File [WSB98], which applies a sequential scan of the database
and prunes single objects according to a high-speed filter distance based on
data quantization and precomputed distances.

Indexing structures based on compact information are mostly hierarchical
in nature, and the compact information correspond to bounding regions,
either implicit or explicit. The minimal distance of query to a bounding
6.2. Efficient Query Processing

Figure 6.2: R-Tree illustration for a 2-dimensional vector space: A hierarchical nesting of minimal bounding rectangles enables pruning of the search space. On the right, a range query is shown which only has to access one of the leaf nodes.

region (and thus to the contained objects) is used to prune the whole region with its contained objects from the search space.

Metric indexing structures (e.g., M-Tree [CPZ97]) rely on the triangle inequality of the applied distance function to prune object groups from the search space. In such structures, bounding regions are solely based on distance information between objects, and not on the actual values of the objects in their respective dimensions (in case dimensions exist at all). An object subgroup is characterized by a pivot element; the maximum distance of data objects in the subgroup to the corresponding pivot element establishes an (implicit) bounding sphere.

The most prominent example for multidimensional indexing structures with explicit bounding regions is the R-Tree family [Gut84, BKSS90]. These trees are balanced and data objects are contained in the leaf nodes. They were developed for spatial data and condensation is obtained by hierarchically nested Minimal Bounding Rectangles (MBR), as illustrated in Figure 6.2. An MBR represents a subtree and encloses all data points contained in that tree; that is, for an object group $O$ in a $d$-dimensional space, the MBR is defined as

$$MBR(O) = ([\min_{o \in O}\{o_1\}, \max_{o \in O}\{o_1\}], \ldots, [\min_{o \in O}\{o_d\}, \max_{o \in O}\{o_d\}]).$$

During query processing, the minimal distance $\text{minDist}$ from the query point to an MBR is used to prune the corresponding subtree from the search space;
accordingly, a $\text{minDist}$ function can be interpreted as lower-bounding filter for a whole subtree (i.e., an object group). In the following chapters, several examples for $\text{minDist}$ functions will be given.

While R-Trees are highly suitable for lower-dimensional settings, their performance degrades in higher dimensional features spaces [WSB98]. This effect is subsumed under the term Curse of Dimensionality, and in case of R-Trees it can be caused by an increasing overlap of MBRs. Several new variants of the R-Tree were developed to cope with this issue. We will discuss this in detail in the following chapter, where we develop an indexing structure for the Euclidean distance. It will serve as an example for an effective integration of the query processing concepts introduced in this section, i.e. filter-and-refine and multidimensional indexing. It is designed for high dimensional complex data, and a key contribution of this indexing structure is filter-and-refine in which each object has an individual set of filters.

Part IV of the thesis will cover the specific topic of time series similarity search, where we contribute new techniques for highly-efficient query processing under a complex time series distance function.
Chapter 7

Subspace Clustering for Efficient Query Processing

∗Fast similarity search in high dimensional feature spaces is crucial in today’s applications. Since performance of traditional index structures degrades with increasing dimensionality, concepts were developed to cope with this curse of dimensionality. Most of the existing concepts exploit global correlations between dimensions, but correlations are often locally constrained to a subset of the data and every object can participate in several of these. Discarding the same dimension set for each object based on global correlations and ignoring the different correlations of single objects leads to significant loss of information. The degree of information preserved, however, has a direct influence on the achievable query performance.

We introduce a novel main memory indexing structure with increased information content for each single object compared to a global approach. The structure of our index is based on a multi-representation of objects reflecting their multiple correlations; that is, besides the general increase of information per object, we provide several individual representations for each single data object. These multiple views correspond to different local reductions per object and enable more effective pruning.

∗This chapter has been published in the Proceedings of the 14th International Conference on Extending Database Technology (EDBT/ICDT 2011) [GKLS11b].
7.1 Motivation

Index structures were introduced for efficient similarity query processing in multidimensional feature spaces. The structures are mostly based on hierarchically nested minimal bounding regions. With increasing feature space dimensionality or information content, many of the first solutions, e.g. the R-Tree [Gut84], are inapplicable for the given tasks [BGRS99, HKK+10, FK94, WSB98]. This is often denoted as the curse of dimensionality: with increasing dimensionality or information content the index performance degrades, eventually becoming slower than a sequential scan of the database.

Filter-and-refine frameworks as well as revised and new index structures were introduced to weaken the effects of high dimensionality. These approaches perform dimensionality reduction by exploiting global correlations between the feature space dimensions corresponding to redundancy in the data. Exploiting global correlations weakens the curse of dimensionality, but the general problem is still existing. The reason is the complex structure hidden in the data: specific correlations between the dimensions are often local, i.e. they are constrained to subsets of the data, and every object in the data can participate in several of these correlations. Reductions based on global correlations, however, cannot reflect the local correlations in an advantageous way for index construction; all objects in the data will be reduced to the same dimensions, and hence many dimensions are discarded that are important for specific subsets of the data. In the remaining dimensions the values in these subsets have a high variance preventing compact minimal bounding regions in the index and resulting in unnecessary large query durations.

Figure 7.1 shows an example, with two coordinate systems representing a 4-dimensional feature space. Examining all objects together, there is no correlation between the dimensions that enables a reduction beneficial for index construction; i.e., finding patterns in a single reduced space allowing the construction of compact minimal bounding regions is not possible. When only subsets of the data are considered, however, it contains several good patterns marked as $C_1$, $C_2$, $C_{1a}$, and $C_{1b}$. For each of the corresponding
7.1. Motivation

Figure 7.1: Hierarchically nested local reductions.

object sets a sound reduction can be found corresponding to a local dimensionality reduction. Constructing an index based on such local reductions is more effective: dimensions with high variances in the value distributions are avoided and more compact bounding regions are achieved resulting in faster query processing due to earlier pruning of candidates.

Accordingly, for complex data indexing frameworks are needed whose filter steps are based on local reductions. One possible solution is to build an index for every local pattern in the data, i.e. every index corresponds to a different filter with another set of reduced dimensions [CM00, SZZ07]. These approaches have a serious drawback: They do not consider that every object can participate in several local reductions, i.e. every object is only assigned to one of these reductions. The potential for several different filters for each data object and better pruning based on these filters is wasted.

In this chapter, we introduce a novel index structure that is based on local reductions and that in particular regards the multiple local correlations of single data objects. This is achieved by building a hierarchy of local reductions that corresponds to the structure of the index tree; that is, a local reduction-based minimum bounding region can itself contain local reductions that enable compact bounding regions in lower levels of the index tree. In contrast to existing approaches, our single index corresponds to a series of different filters for every data object; every query passes through an individual cascade of filters until it reaches a leaf node.
Technically, we realize these new ideas by transferring methods from data mining: we detect patterns which are used to construct a better index. For detecting data subsets combined with a set of dimensions suitable for local reduction, we use subspace clustering. A subspace cluster is a grouping of objects that are only similar in a subset of their dimensions and thus minimum bounding regions induced by subspace clusters are very compact. The underlying structure of our index structure is a hierarchy of nested subspace clusters. For example, in Figure 7.1 our approach can identify the illustrated hierarchical nesting. There are interesting implications: Due to the nature of subspace clustering, it is possible that subspace clusters on deeper levels have other relevant dimensions than their parent clusters. This corresponds to a multi-representation of objects, i.e. on each level objects can be represented by a different set of dimensions defining individual local reductions.

Clustering algorithms are often unstable, i.e. they are parameter-sensitive and every run delivers different results; therefore we introduce a method that is motivated by train-and-test, a paradigm that is well established in the data mining and machine learning domain, e.g. for decision tree construction. In applications where the underlying distribution of incoming data objects changes, our index adapts its underlying clustering structure dynamically.

Many existing index structures are designed for secondary storage, and thus technical constraints as block size must be adhered to. In many domains, however, this is becoming obsolete: Main memory capacity of computers is constantly increasing and often persistence of the index structure or even the dataset has lost importance compared to the need for fast responses. An example are stream classification scenarios. For main memory indexing, random I/O is no longer a problem allowing the development of more flexible index structures. For example, our unbalanced index structure can reflect the inherent structure of the data better than structures that are constrained by balance properties: different parts of the dataset can be represented in different granularities, allowing more efficient access.

The chapter is structured as follows: Section 7.2 discusses related work on indexing techniques. Section 7.3 introduces our new approach. Section 7.4 presents the evaluation and Section 7.5 concludes the chapter.
7.2 Comparison with Related Work

In this section we discuss related work on indexing and subspace techniques.

**Indexing techniques.** Techniques for similarity search can be categorized into approximate and exact solutions. Methods from the former category trade accuracy for speed, as for example Local Sensitive Hashing [TYSK09, GIM99]. Methods from the latter category produce exact results, i.e. there are no false dismissals or false positives in the result set. Since our proposed index is from this category, we focus on exact solutions in the following.

The R-Tree [Gut84] is one of the first multidimensional indexing structures. It is well suited for lower dimensions, but its performance rapidly degrades for higher dimensionalities due to the curse of dimensionality; a main cause is the presence of irrelevant attributes, as demonstrated in [HKK+10]. To mitigate the curse, global techniques like PCA [Jol02] or cut-off reduction are used to reduce the dimensionality of indexed objects. The drawbacks of global dimensionality reduction were discussed in Section 7.1. Several indexing structures for high dimensional feature spaces, mostly based on the R-Tree, were proposed. The R*-Tree [BKSS90] introduces new split and reinsertion strategies. The X-Tree [BKK96] enhances the R*-Tree by introducing overlap-minimizing splits and the concept of supernodes; if no overlap-minimizing split is possible, supernodes of double size are generated, eventually degenerating to a sequential scan. The A-Tree [SYUK00] uses quantization to increase the fan-out of tree nodes. The TV-Tree [LJF94] is based on a PCA-transformed feature space: Minimum bounding regions are restricted to a subset of active dimensions in this space. Active dimensions are the first few dimensions allowing for discrimination between subtrees. It has, however, the same drawbacks as the other global reduction techniques. The metric index iDistance [JOT+05, YOTJ01] realizes distance-based indexing: Data points are transformed into single dimensional values w.r.t. their similarity to specific reference points. These values are indexed by a $B^+$-tree and the search is performed by one-dimensional range queries in these tree.

All of the described index structures are optimized for secondary storage, i.e. they have node size constraints. None of these index structures
accounts for local correlations in the data. An approach that makes use of local correlations is LDR [CM00]. It uses subspace clustering to create a single clustering of the whole dataset. These clusters represent local correlations. Dimensionality reduction is performed individually on the clusters and an index is built for each of the reduced representations. LDR has several drawbacks: it applies subspace clustering only to the root of the constructed tree, while the subtrees are built the conventional way (one index for every subtree). These indices are still prone to the curse of dimensionality. Another local-correlation based approach is MMDR [SZZ07]. In difference to LDR, a single one-dimensional index is used. In both LDR and MMDR, there is no hierarchical nesting of subspace clusters, i.e. there is no multi-representation of objects according to different dimensions and there is no handling of local correlations in the used indices.

Subspace techniques. Subspace clustering was introduced in Chapter 2. Besides the huge amount of algorithms that keep the original dimensions, some algorithms [AY00] transform the data space on detected correlations. Initial work has also be done in detecting hierarchies of subspace clusters [ABK+06]. However, the computational complexity of the approaches avoids efficient application. Another concept is subspace similarity search [BEG+10], in which queries are processed in subspaces of the data; we concentrate on similarity queries in the full-dimensional space.

7.3 Subspace-Clustering-Based Indexing

In this section, we present SUbSace-based High-dimensional Indexing, in short: SUSHI. In Section 7.3.1, we motivate and define the node structure of the tree. Index construction considering static and dynamic usage is presented in Section 7.3.2. Section 7.3.3 presents the query processing strategy.

7.3.1 Tree structure

The general idea of our index structure is to represent each object in multiple ways, such that different information can be used for pruning. To enable efficient query processing we partition the data in a hierarchical structure,
allowing the pruning of whole subtrees if they are not important for the current query. Each subtree represents a subset of objects annotated with its local reductions avoiding the information loss obtained by global reduction. To determine these local reductions we use subspace clustering methods, since in high dimensional data we cannot find meaningful partitions with traditional approaches due to the curse of dimensionality [HKK+10, BGRS99].

Unlike existing indexing approaches we recursively apply subspace clustering on smaller subsets of the database, realizing multi-representations for each object. We identify the local correlations in the whole database but also on more fine-grained views of the data. In Figure 7.1, for example, the subspace cluster \( C_1 \) in dimensions 1 and 2 can further be refined by the clusters \( C_{1a} \) and \( C_{1b} \). We can identify these locally nested patterns with subspace clustering on the object set of \( C_1 \).

Formally, a subspace cluster is a set of objects together with a set of locally relevant dimensions. The objects show high correlations and thus compactness within the relevant dimensions while in the irrelevant ones we cannot identify a good grouping for this set of objects. For different groups of objects different relevant dimensions are possible, thus we are not constrained to a global reduction of the data. A subspace clustering is a list of subspace clusters together with a list of outliers. The handling of outliers is important because not every object shows a good correlation to other objects. If we prohibited outliers, the clustering quality would degrade and we could assume that also our index shows poor performance.

**Definition 7.1** Given a set of dimensions \( \text{Dim} \) and a database \( \mathcal{D} \subseteq \mathbb{R}^{\mid\text{Dim}\mid} \), a **subspace cluster** \( C \) is defined by \( C = (O, S) \) with objects \( O \subseteq \mathcal{D} \) and relevant dimensions \( S \subseteq \text{Dim} \). A **subspace clustering** \( \text{Clus} \) is defined by

\[
\text{Clus} = (C_1, \ldots, C_k, \text{Out})
\]

with clusters \( C_i = (O_i, S_i) \) \((i = 1, \ldots, k)\) and outlier list \( \text{Out} = \mathcal{D} - \bigcup_{i=1}^{k} O_i \).
Each object $o \in DB$ is either in a cluster or in the outlier list. We use subspace clustering methods which partition the data in each step, i.e. the sets $O_i$ are pairwise disjoint. This is reasonable for constructing an index structure, because otherwise we would include a single object in multiple nodes (on different paths in the tree) leading to a hindered pruning of this object. Apart from that, our index structure is independent from the underlying clustering algorithm. New developments in the research area of subspace clustering can directly be applied to our index. In Section 7.3.2 we describe the used algorithms.

Index structures are based on the idea of pruning parts of the search space. In our index structure, each node is determined by a subspace clustering containing several clusters. To enable pruning of all the objects within a cluster we have to condense a cluster $C = (O, S)$ to some compact information, i.e. a minimum bounding region: we use the idea of enclosing rectangles in the relevant dimensions of the subspace cluster, i.e. each cluster is represented by lower and upper bounds in its subspace.

**Definition 7.2** Given a subspace cluster $C = (O, S)$, the **Subspace Enclosing Rectangle (SER)** $R$ is a list of lower and upper bound values for the relevant dimensions of the cluster, i.e. $R = ([i_1, low_1, up_1], \ldots, [i_d, low_d, up_d])$ with $\{i_1, \ldots, i_d\} = S$ and $low_j = \min_{o \in O} \{o_{i_j}\}$, $up_j = \max_{o \in O} \{o_{i_j}\}$, where $o_{i_k}$ is the restriction of $o$ to dimensions $k$.

The irrelevant dimensions of a subspace cluster provide no or little information about the clustered objects. We do not store this information, which corresponds to a local dimensionality reduction and leads to reduced computational effort during query processing. We focus on this simple representation to analyze the effect of hierarchically nested subspace clusters but not to figure out the best cluster approximation. Other approximations could be included in our index, as elliptical representations or transformation-based approaches; however, in Section 7.3.3 we discuss why these methods are no good choice.

In SUSHI we distinguish between inner nodes and leaf nodes. The root node represents the whole database $DB$. 
Definition 7.3 An inner node $N$ representing the objects $O$ fulfills:

- $N$ is determined by a subspace clustering $\text{Clus} = (C_1, \ldots, C_k, \text{Out})$ for the set $O$.
- For each cluster $C_i = (O_i, S_i)$ the node $N$ contains a SER $R_i$.
- For each SER $R_i$ the node $N$ contains a pointer to its child node representing the subset $O_i$.
- $N$ stores a list of objects (or pointers) corresponding to $\text{Out}$.

A leaf node $N$ representing the objects $O$ fulfills:

- $N$ stores a list of objects (or pointers) corresponding to $O$.

In the leaf nodes full-dimensional objects $o \in DB$ are stored, and each inner node can also contain a list of full-dimensional objects corresponding to the outliers obtained by one clustering step performed on the respective subset of objects. Thus we are able to identify local outliers, i.e. outliers that emerge only by regarding a subset of objects. The potential to store objects also in inner nodes yields compact representations of the remaining clusters. Otherwise, the enclosing rectangles would grow disproportionately large to account for the outliers.

Each subspace cluster within an inner node but also across several layers of the tree can have a different set of relevant dimensions. We do not cut off some dimensions globally, and we do not represent all objects in the same way; thus we do not perform the well-known filter-and-refine query processing [FRM94] that is normally based on a single filter step. Instead, each object is represented in different ways starting from the root down to its full-dimensional object description. Each object has its individual multirepresentation enabling us to prune objects along one path of our tree according to their various representations. Thus our index structure has inherently realized the filter-and-refine approach.

In Figure 7.2 the tree structure is outlined. The root node is determined by three subspace clusters of different sizes. Adjacent to each cluster the relevant dimensions are visualized by green squares, e.g. the first subspace
cluster is located in the subspace \( \{2,5\} \). The objects are reduced to this local
dimension set. Each cluster is represented by a subspace enclosing rectangle
that is stored in the node. Besides the SERs, the root node stores a list of
outliers; in this example with three objects. Each cluster can either split up
in further clusters, as for example the first cluster of the root, or a leaf node
is appended as for the root’s second cluster. Subspace clusters in different
levels of the tree can represent different sets of dimensions, realizing multi-
representation of objects.

Our index is a main memory structure, hence it is not restricted by block
sizes. In main memory, the time to access a child node is negligible compared
to secondary storage. This gives us the possibility to construct unbalanced
trees and some objects or regions can be represented more detailed. A couple
of objects have many different representations while others have only few.

![Figure 7.2: Tree structure of SUSHI.](image)

### 7.3.2 Tree construction

For index construction, we distinguish two cases: First, static construction,
where we build the index for a database provided in whole. Second, dynamic
construction where we cope with insertion and deletion of objects.
Static construction

Usually an initial set of data is available to construct our index via bulk-loading. To achieve the multi-representation of objects we need a subspace clustering for each inner node of the tree. A simple approach is to calculate a single clustering for the root node and recursively repeat this procedure for each subset to construct further nodes. This procedure can lead to instabilities during the index construction: Since there exist several clusterings for a single dataset – each clustering with different quality for grouping the objects – it is unlikely to get a good result with a single run of a clustering algorithm. Furthermore many algorithms are non-deterministic and they inherently construct different clusterings. In the worst case we get high variations for the index construction and, thus, query performance could deviate to a high degree. To improve stability and quality, we calculate multiple clustering per node and select the one with highest pruning potential. The pruning potential of clusterings is a key concept in our indexing structure and is described in a separate paragraph.

Since in our hierarchical structure the child nodes depend directly on the upper level clustering, we use the following approach to construct the index. We start with the root for which multiple clusterings are calculated. Consequently, we get a set of partly constructed models/index structures. We select the best model from this set and we try to complete it by selecting one cluster from the model’s root whose corresponding object set is clustered again several times. Based on our first partly constructed model we get a new set of extended but still not complete index structures. The best index is selected and the approach is recursively applied until the number of objects in a node is below a threshold $\text{minSize}$; in this case, a leaf node is constructed.

Summarized: At each point in time we extend our index structure with one inner node. This inner node is determined based on a set of inner node candidates, i.e. subspace clusterings, and in each step the clustering with highest pruning potential is selected.

**Pruning potential of clusterings.** To determine this best clustering we use an approach inspired by the train-and-test method used in other data
mining applications. On the one hand, one can use train-and-test to evaluate the quality of an approach. On the other hand, train-and-test is used to construct better models itself, e.g. for decision trees [Mit97]. The general idea is to build a set of models based on the training set and choose the best one with respect to the test set. In the literature several measures to judge the quality of clusterings, e.g. the compactness, are presented [HBV01]. However, in our approach the clustering with the highest quality needs not to be equivalent to the best node for the tree. In our approach, we seek for the clustering resulting in the best query performance for our index, i.e. the clustering with highest pruning potential. Motivated by this we measure exactly those costs that are important during the utilization of the index: the distance calculations (to enclosing rectangles and points) needed to determine, e.g., a nearest neighbor. Since we start our index construction at the root node, each partly constructed index is a valid index structure with respect to Definition 7.3. Thus, we can temporarily append a new inner node and check how many distance calculations are performed to find the nearest neighbors of the test set’s objects. Then the subspace clustering determining the best inner node based on this objective function is selected.

Selecting the right test set is important while evaluating our objective function. In an ideal case the data distribution of the test set follows the distribution of possible query objects. Since this distribution is unknown we assume that the database itself reflects this distribution. Hence our test set is randomly and uniformly sampled from the whole database $\mathcal{DB}$. Although we optimize our tree locally (‘what is the best new node for one subtree’), we should not use a sample of the current subtree as the test set. In this case the test set would not be representative and the subtree is not optimized with respect to the performance of the whole tree. To avoid overfitting and enhance the generalization we generate a new test set in each recursive step (i.e., for each new node to be constructed).

Algorithm 7.1 describes the overall construction of nodes. To generate the root node, the whole database $\mathcal{DB}$ is the input. During the hierarchical data partitioning, the object subsets get smaller. If the number of objects for a node is below a certain threshold we create a leaf node (lines 2-4).
7.3. Subspace-Clustering-Based Indexing

Algorithm 7.1: Static node construction algorithm.

```
1 input: set of objects $O$;
2 if $|O| < \text{minSize}$ then
3     constructLeafNode(); // based on $O$
4     return;
5 Test = chooseTestSet();
6 // initial clustering
7 bestClust = performClusteringOn($O$);
8 bestQuality = evaluateClust(bestClust, Test);
9 stableCount = 0;
10 while stableCount < stableSteps do
11     currClust = performClusteringOn($O$);
12     currQuality = evaluateClust(currClust, Test);
13     if currQuality > bestQuality then
14         bestClust = currClust;
15         bestQuality = currQuality;
16         stableCount = 0;
17     else stableCount ++;
18 constructInnerNode(); // based on bestClust
19 foreach $C_i = (O_i, S_i)$ of bestClust do
20     run node construction for $O_i$;
```

The train-and-test method starts with the lines 5-7. We choose the test set, perform a subspace clustering on the objects of $O$ and evaluate the quality of this clustering with respect to the test set. In lines 9-16 we try to enhance the clustering quality. Based on the best evaluated clustering we construct the inner node (line 17). At last, the node construction method is called recursively for each cluster (line 19) to realize the hierarchical structure.

Subspace clustering algorithms. Our index is able to utilize arbitrary subspace clustering algorithms. In a recent evaluation study of subspace clustering [MGAS09], PROCLUS [APW+99] and MineClus [YM03] performed best. Both methods have low runtimes and they do not generate arbitrarily widespread clusters but compact approximations, making them highly suitable for our index construction. PROCLUS extends k-means to subspace clustering and assigns points to its nearest representative. MineClus uses
(subspace) rectangles of a certain width to identify dense regions, and these boxes approximate the clusters. MineClus is a deterministic algorithm, while PROCLUS is non-deterministic; i.e., assuming the same object set and fixed parameters, the former generates the same clustering in each run, while the latter can generate varying clusterings. Accordingly, only PROCLUS is suited for our train-and-test method, which selects the best clustering from a given set of different clusterings for the same object set. MineClus is not directly applicable with train-and-test, but we still can generate one clustering per node. We analyze this difference in the experiments.

**Dynamic construction**

In the following, we consider the insertion of new objects and the deletion of existing objects.

**Insertion.** The insertion method makes use of an already existent SUSHI index. First, we identify the leaf node whose enclosing rectangle (which is included in the node one layer above the leaf) shows the smallest $\text{minDist}$ with respect to the object we want to insert. The objects in this leaf node are good candidates for clustering with the new object. To assess if this leaf node is truly a good candidate for insertion we check whether the volume of the enclosing rectangle does not increase too much (i.e., an increase of 10% in volume is allowed). Otherwise, we recursively go one layer up in the tree and test the corresponding rectangle. Instead of a leaf node we now analyze inner nodes, thus the object would be inserted in an outlier list. Evidently, all enclosing rectangles including the newly inserted object must be updated to retain a correct approximation of the clusters.

With a reclustering step we ensure that our index adapts to new data distributions. We monitor the number of outliers in a node and if this number increases too much (more than twice of the list’s initial size), we rearrange the subtree with the methods presented in the dynamic construction section.

**Deletion.** To delete an object we have to identify the path from the root to the leaf or inner node where the object is stored (the object could be an outlier and thus stored in an inner node). The object is removed from
the identified node and all enclosing rectangles up to the root are possibly downsized. In the construction phase (cf. Algorithm 7.1) we do not split up a cluster if the number of objects is below a certain threshold. Vice versa, for the deletion, we have to identify the cluster on the highest level of the currently considered path with too few objects (i.e., less than \( \text{minSize} \)). Its complete subtree is removed and substituted by a leaf node. This procedure prevents that long paths with very small clusters are maintained, which would incur large processing times.

### 7.3.3 Query processing

We focus on \( k \)NN processing with Euclidean distance but other types can be easily integrated. In Section 7.3.1 we approximate the subspace clusters by subspace enclosing rectangles (SERs). To ensure completeness of our index, i.e. no false dismissals are allowed, we have to define a \( \text{minDist} \) that has to be a lower bound for all objects within the underlying subtree: the distance from the query object \( q \) to the compact information is smaller than the distance to each object in the subtree.

**Definition 7.4** The Minimum Distance \( \text{minDist} \) between query \( q \in \mathbb{R}^{|\text{Dim}|} \) and the SER \( R = ([i_1, low_1, up_1], \ldots, [i_d, low_d, up_d]) \) is defined as:

\[
\text{minDist}(q, R) = \sqrt{\sum_{j=1}^{d} \begin{cases} (low_j - q_{i_j})^2 & \text{if } q_{i_j} < low_j \\ (up_j - q_{i_j})^2 & \text{if } q_{i_j} > up_j \\ 0 & \text{else} \end{cases}}
\]

Algorithm 7.2 gives an overview of the query processing. It is based on the \( k \)NN optimal processing approach in [SK98]. Starting with the root node, a priority queue stores the currently active nodes. At each point in time we refine the node with the smallest \( \text{minDist} \) (l. 7). As mentioned before, in each node we are able to store full-dimensional objects. These objects are either outliers or objects from leaf nodes (l. 8 or 10) for which we perform a linear scan to update the temporary nearest neighbors (l. 12-16). Additionally, if the current node is an inner node we add its child nodes to the queue.
Algorithm 7.2: kNN queries in SUSHI.

1 input: query \( q \), result set size \( k \)

2 \( queue = \text{List of } (\text{dist, node}) \) in ascending order by dist;

3 \( queue.\text{insert}(0.0, \text{root}); \)

4 \( resultArray = [(\infty, \text{null}), \ldots, (\infty, \text{null})]; \) // \( k \) times

5 \( dist_{\text{max}} = \infty; \)

6 while \( queue \neq \emptyset \) and \( queue.\text{nextDist} \leq dist_{\text{max}} \) do

7 \hspace{1em} \( n = queue.\text{pollFirst}; \)

8 \hspace{2em} // scan objects of leaf or possible outliers

9 \hspace{2em} if \( n \) is leaf node then \( toScan = n.O; \)

10 \hspace{2em} else \( toScan = n.\text{Out}; \)

11 \hspace{2em} foreach \( o \) in \( toScan \) do

12 \hspace{3em} if \( \text{dist}(q, o) \leq dist_{\text{max}} \) then

13 \hspace{4em} resultArray\[k\] = (\text{dist}(q, o), o);

14 \hspace{4em} resultArray.\text{sort};

15 \hspace{4em} dist_{\text{max}} = \text{resultArray}[k].\text{dist};

16 \hspace{2em} if \( n \) is inner node then

17 \hspace{3em} foreach \( \text{SER} R \) in \( n \) do

18 \hspace{4em} if \( \text{minDist}(q, R) \leq dist_{\text{max}} \) then

19 \hspace{5em} queue.\text{insert}(\text{minDist}(q, R), \text{R.child});

21 return \( resultArray; \)

(l. 17-20). The sorting is based on their \( \text{minDist} \) values. It is important to scan the outliers before analyzing the child nodes: the value of \( dist_{\text{max}} \) can be lowered (l. 16) and further subtrees can be pruned.

In Section 7.3.1, we mentioned the possibility of using transformation-based approaches for cluster approximation. Accordingly, the \( \text{minDist} \) calculation would also be based on this transformation. However, due to our multi-representations we would use different transformations for each node / cluster and hence the query object has also to be transformed several times (necessary for line 19) resulting in inefficient processing. Therefore we concentrate on the method of subspace enclosing rectangles.
7.4 Experiments

This section is structured as follows: Section 7.4.1 describes the setup. Section 7.4.2 studies the different construction strategies and parameters. Finally, Section 7.4.3 compares SUSHI to several competing approaches.

7.4.1 Experimental setup

We compare our SUSHI with index structures from different paradigms. The R*-tree [BKSS90] is used as a competitor for full space indexing. Global dimensionality reduction approaches are realized by using the R*-tree with PCA or Cut-Off reduction: for the first we remove the dimensions with the lowest information content, for the second we simply remove the last dimensions. As an approach exploiting local correlations we implement the LDR index [CM00]. Distance-based indexing with respect to reference points is realized by the iDistance method [JOT+05]. Additionally, the sequential scan is used as a baseline competitor. Since SUSHI is designed for main memory and not for secondary storage, node/page accesses are irrelevant. Instead, as an implementation invariant performance measure, we use the number of distance calculations to the bounding regions and to the data objects/outliers (with equal weight) for all approaches.

We evaluate the performance on several real world and synthetic data sets. We use color histograms in the extended HSL color space (ext. by dimensions for gray values) as features obtained from a dataset that combines well-known image databases (Corel, Pixelio, Aloi, Hemera). We use the UCI pendigits data [FA10] and extend it to a 48-dimensional variant by interpolation of the available polylines. Moreover, a 15-dimensional dataset reflecting oceanographic characteristics as temperature and salinity of the oceans is used†. For synthetic data, we follow the method in [GMFS09, KKK04] to generate density-based clusters in arbitrary subspaces. The generator takes into account that subspace clusters can be hierarchically nested (cf. Figure 7.1).

†provided by the Alfred Wegener Institute for Polar and Marine Research, Bremerhaven, Germany
by allowing a varying subspace cluster hierarchy depth. Unless otherwise stated, we generate data with 10,000 objects, 64 dimensions, 16 clusters in a hierarchy of depth 4, and 5 percent noise (outliers w.r.t. clustered objects) per level of the hierarchy.

For repeatability and comparison we specify the default parameter settings used in our experiments. We measure the number of distance calculations to obtain the 5 nearest neighbors averaged over 100 queries following the data distribution. We set \( \text{minSize} \) (cf. Alg. 7.1) and the number of clusters for PROCLUS to 20. The test set contains 50 objects and \( \text{stableSteps} = 5 \). For the LDR approach, we use \( \text{Minsize} = 10 \) and \( \text{FracOutliers} = 0.1 \) as in the original publication [CM00]. According to [JOT+05], we set the number of reference points in iDistance to 64, and they are determined by k-means. The node size for the R*-tree methods is set to 4kb. All dimensionality related parameters (average dimensionality of cluster in PROCLUS; number of retained dimensions in the PCA and Cut-Off approach; maximal dimensionality of clusters in LDR) were optimized for each dataset such that the best query performance is obtained.

### 7.4.2 Evaluation of construction strategies

We start by evaluating the different construction strategies of SUSHI, i.e. the train-and-test method and the applied clustering algorithms. Furthermore, we study several parameter settings of our index.

**Train-and-Test.** First, we analyze how the train-and-test strategy influences the query performance of the PROCLUS-based version of SUSHI. Fig-
7.4. Experiments

Figure 7.3 compares the efficiency with or without train-and-test for varying average dimensions per subspace cluster, a parameter of PROCLUS. The number of average distance calculations for the variant without train-and-test is averaged over 5 constructed indices, while the optimized variant uses just one index. That is, the results are averaged over 5-100 queries or 100 queries, respectively. This procedure was chosen because the variant without train-and-test is very unstable, i.e. the nondeterminism of PROCLUS creates very different indices in each construction phase. Therefore, the query efficiencies of the different indices vary considerably. In the figure, this is pointed out by the whiskers showing high variances in the numbers of distance calculations. The train-and-test-based method clearly outperforms the other variant. Considering the best-case situations of the variant without train-and-test, i.e. the lower ends of the whiskers, the train-and-test approach, averaged over only one index, dominates over all dimensionalities. Train-and-test significantly improves the stability of the clustering and thus the quality of PROCLUS-based SUSHI. Because of the good results, PROCLUS-based SUSHI is always combined with train-and-test in the remaining experiments.

MineClus vs. Proclus. Next, we study the differences between MineClus and PROCLUS-based SUSHI. Figure 7.4 compares the approaches on two datasets. To give a better overview, we evaluated two different parameters, i.e. the database size for the oceanographic dataset and the dimensionality for the histograms. For MineClus, most of the recommended parameter settings from the original publication [YM03] were applied: $\beta = 0.25$, $\text{maxOuterIterations} = 100$, and $\text{numBin} = 10$. The side length $w$ of a hypercube was automatically determined by a heuristic proposed in [PJAM02] and is based on a sampling set of 2,500 objects. Because we observed a higher performance with lower values of $\alpha$, we fixed it at $\alpha = 0.001$. In both experiments, PROCLUS-based SUSHI shows a substantially higher performance than MineClus-based SUSHI. This is especially the case for higher dimensionalities of the histogram dataset; runtimes of PROCLUS-based SUSHI stay relatively constant with increasing dimensionality, while the performance of MineClus-based SUSHI degrades. The experiments point out that the subspace clusters generated by PROCLUS are better candidates for SERs than
the ones generated by MineClus. For the remaining experiments, PROCLUS-based SUSHI is the method of choice.

Figure 7.5 shows how the different PROCLUS parameters influence the performance of SUSHI. In these experiments, the dataset (pendigits) is fixed. The studied parameters are the average dimensionality of subspace clusters and the number of clusters to be found.

Average dimensionality. The experiment in Figure 7.5(a) shows that the query performance of SUSHI is highly influenced by this PROCLUS parameter, which corresponds to the average dimensionality of the generated subspace clusters. Most interesting, with higher dimensionalities (≥ 20) query performance rapidly deteriorates; we can infer that the usage of local correlations in the data can improve the performance of our index significantly.
Accordingly, higher average dimensionalities should not be used as parameters. Furthermore, very low average dimensionalities are also no good parameter setting. There is not enough information in such low-dimensional subspace clusters for creating an efficient index.

**Number of subspace clusters.** The experiment in Figure 7.5(b) displays how the query performance is influenced by the number of clusters to be found. The general tendency is that with a higher number of clusters the average number of distance calculations becomes more stable, i.e. the results for a low number of clusters are very fluctuant and show no good query performance. As a result, parameterizing PROCLUS with a low number of clusters should be avoided. The effect is, however, very dataset dependent. Based on several experiments with different datasets we made a trade-off and selected a cluster number of 20.

**Influence of dynamic inserts.** The experiment in Figure 7.6 studies how dynamic inserts influence the query performance. The x-axis describes to which percentage the index is based on dynamic inserts. Before the dynamic inserts, the rest of the data objects is inserted statically, a process known as bulk-loading. Accordingly, the database size is always fixed. The dataset is pendigits with 10,992 objects. Overall, the results are very stable. That is, the benefits of bulk-loading the index are negligible. Only if the whole index is built with bulk-loading (i.e., 100% static), a slight performance increase can be noticed. The overall good performance can be explained by the insert strategy based on dynamic reclustering. It adapts to new data distributions making SUSHI well suited for dynamic application domains.
7.4.3 Comparison with competing approaches

In the following experiments we compare the query performance of SUSHI to the ones of other approaches.

Database size. In Figure 7.7(a) we evaluate a varying database size by using subsets of our color histogram database. SUSHI requires very few distance calculations and the slope of the curve is small. All competing methods show a higher increase with respect to the database size. Especially the LDR method, which is able to use local correlations, shows worse performance than the PCA-based method, which detects only global ones. In this experiment, the R*-tree with and without Cut-Off reduction show the same poor performance. This is due to the characteristics of the dataset: cutting off the last dimensions corresponds to discarding the gray value dimensions of the extended HSL color space. These dimensions, however, have high discriminative power due to their rich information content. Accordingly, best performance is achieved by not cutting off any dimensions.

In Figure 7.7(b) we analyze the effects of varying database size on a synthetic dataset. Comparable to the first experiment, SUSHI clearly outperforms all other approaches. In this diagram, we skip the sequential scan corresponding to a diagonal for clarity. The LDR approach cannot cope with the hierarchy of subspace clusters, it uses subspace clustering only once. The performance degenerates to the one of the R*-tree or is even worse. Furthermore, using classical index structures after the clustering step within the LDR method leads to similar problems as for the R*-tree.
7.4. Experiments 139

Figure 7.8: Varying dimensionalities and degree of noise on synth. data.

**Dimensionality.** In Figure 7.8(left) we increase the dimensionality of the dataset. For SUSHI and iDistance no increase in the number of distance calculations is observed, while all the other approaches have increasing numbers of calculations. Even for the 96-dimensional dataset the efficiency of our approach is high. The characteristics of LDR and the R∗-tree methods are similar to the previous experiment. The sequential scan constantly needs 10,000 calculations and is orders of magnitude slower than SUSHI.

**Noise.** Now we analyze the effects of an increasing percentage of noise in the data. Noise is present in nearly all datasets and the index structures should handle this as well. Figure 7.8(right) demonstrates the strength of SUSHI. While its processing time stays low, some competing approaches even converge to the sequential scan. Our SUSHI is able to store outliers, i.e. noise objects, separately in the inner nodes of the index; thus, our subspace enclosing rectangles are not influenced by the outliers and we reach compact representations of the objects. In contrast, the R∗-tree must include all outliers, and in iDistance the selection of reference points is sensitive to noise, leading to poor performance of both paradigms. Again and unlike SUSHI, LDR cannot identify the cluster hierarchy.

**Information content: Depth of subspace cluster hierarchy.** It was shown that high dimensionalities do not necessarily cause the curse of dimensionality but the information content in the data itself [FK94]. We show that our SUSHI handles this information-rich data better than the other approaches, because we identify and use more structure from the data like hierarchically
nested clusters. Measuring the information content of data can be done multifaceted, e.g. in factor analysis one uses the eigenvectors of a covariance matrix to represent factors and the eigenvalues as indicators for the explained variances by each of these factors [Jol02]. Large eigenvalues correspond to factors explaining an important amount of the variability in the data. A dataset with low information content has few large eigenvalues and several small ones, and the data can be represented by only few factors. In contrast, data with high information content can result to almost equal eigenvalues. Each factor is important to explain the data.

Figure 7.9(a) visualizes the information content for several datasets (one curve per dataset). Each dataset consists of 64 dimensions and hence 64 eigenvalues are calculated. On the x-axis the number of used eigenvalues/factors to explain the data are presented. The eigenvalues are sorted in descending order such that the factors which explain most of the variances are considered first. The y-axis shows the percentage of the explained variances w.r.t. the overall variance. For example, if we use all factors (right side of the diagram) we can explain the whole data (100%). If we use only two factors we can explain only a part of the variance, e.g. 20%. A dataset with low information content reaches nearly 100% with only few factors, while a dataset with high information content tends to be a diagonal in the diagram.

To be concrete, in Figure 7.9(a) we generate synthetic datasets with varying hierarchy depth for nested subspace clusters. If the hierarchically nesting
7.4. Experiments

Figure 7.10: Semi-real glass data with varying dimensionality (shifted by $i$)

is only of depth 4 (left curve), the data contains low information. If we increase the nesting of subspace cluster, the curves slide to the right and hence we increase the information content.

In Figure 7.9(b) we analyze the performance of SUSHI and its competitors on these datasets. Dimensionality and database size are fixed, and we only increase the information content. As expected the performance of all methods drops; however, SUSHI outperforms all other approaches because it can identify the hierarchies in the data and we use this information for pruning. All R*-tree variants degenerate and show even worse performance than the sequential scan. The curse of dimensionality becomes apparent. The LDR methods reveals acceptable results only for small hierarchy depths but converges fast to the sequential scan. The method cannot identify meaningful structures and hence nearly all objects are included in the outlier list which results in a sequential scan behavior. Similarly, iDistance fails on these complex datasets since the indexed distances are no longer discriminable and all objects need to be processed.

Information content: Real world data. Since we cannot directly influence the information content of real world data we modify the dataset and we obtain semi-real data. In the next experiment, our method for varying information content is the following: Assume a database $DB$ with $d$ dimensions is given. To increase the dimensionality by $x$ we replicate the database $x + 1$ times. The new database $DB'$ consists of $x + 1$ different instances $DB_i$. 

![Figure 7.10](image-url)
Thereby, each $DB_i$ is obtained from the old $DB$ by shifting all attribute values of the objects $i$ dimensions to the right, as illustrated in Figure 7.11. The gray shaded cells are filled with random values. Please note that the dimensionality and the database size is modified by this procedure. In our experiments, we use the UCI glass data [FA10] to create our semi-real data. We generate different datasets starting with 10 dimensions (glass10) up to 17 dimensions (glass17).

Just as in Figure 7.9(a), Figure 7.10(a) shows the explained variance by the used eigenvalues/factors. Indeed our method yields an increase in information for the datasets. In Figure 7.10(b) the efficiency of the indexing approaches on these datasets is presented. As in the previous experiment SUSHI yields the highest performance. Keep in mind that the dimensionality and database size increase simultaneously and hence the sequential scan does so too. Anyhow, the slope of the curve for SUSHI is very small. An interesting observation is that with increasing complexity of the dataset the PCA approach performs worse compared to the classical R*-tree.

In the following experiment we change the information content by a different method: Instead of incrementally shifting the replicated databases by one dimension, we directly shift the databases by a factor of $d$. Thus, each $DB_i$ is obtained from the old $DB$ by shifting all attribute values of the objects $d \cdot i$ dimensions to the right. By this method, the dimensionality of the obtained semi-real data increases much faster compared to the previous method. In the experiment we generate data starting with the dimensionality of the original glass data (100%) up to a factor of 400%.

![Figure 7.11: Generation of semi-real data.](image-url)
7.4. Experiments

Figure 7.12: Semi-real glass data with varying dimensionality (shifted by $d \cdot i$)

(a) Variance vs. used eigenvalues

(b) Performance

In Figure 7.12(a) the explained variances by the used eigenvalues/factor are depicted. As illustrated, this data generation procedure also increases the information content of the data. Figure 7.12(b) shows the corresponding efficiency of the indexing approaches on these datasets. Similar to the previous experiments, SUSHI performs best on datasets with a high amount of information. All competing approaches show higher numbers of calculations and most of them quickly converge to the poor performance of the sequential scan. A special case is the iDistance, which performs even worse than the sequential scan.

Number of clusters. In Figure 7.13 we evaluate the query performance when the number of subspace clusters in the data is increased, thus yielding an increase in the local correlations of the data since each cluster has its individual relevant dimensions. Our SUSHI can detect these local correlations

Figure 7.13: Varying number of clusters on synthetic datasets.

Figure 7.14: Performance for different $k$NN queries.
resulting in the highest efficiency. The global dimensionality reduction PCA exhibits a high number of calculations, even worse than the classical R*-tree. With increasing number of subspace clusters global correlations become negligible and cannot be used for effective pruning.

**Number of nearest neighbors.** The performance of the methods under varying result size for the $k$NN query is studied in Figure 7.14. SUSHI shows a nearly linear behavior while the other approaches rise faster. The LDR even degenerates to the poor performance of the sequential scan. Some index structures keep up with SUSHI for the 1-NN query. The potential of SUSHI, however, becomes apparent for nearest neighbor queries with larger result size. These queries are more relevant in practical applications.

### 7.5 Conclusion

In this chapter, we introduced SUSHI for indexing high dimensional objects. Our novel model uses subspace clustering to identify local reductions that achieve higher information content than global reductions. By a hierarchical nesting of local reductions we generate a multi-representation of objects, so that queries traverse a cascade of different filters in the index. Our index construction is optimized via a train-and-test method that provides compact descriptions for regions in the feature space, and we ensure that our index adapts to new data distributions in case of concept drift. Thorough experiments on real and synthetic data demonstrated that SUSHI enables fast query processing and reliably outperforms existing approaches. In the discussion of future work (cf. page 215), we will discuss how the insights gained in this chapter could be used to develop an indexing structure for time series data.

This chapter finishes the general introduction to distance-based similarity search and concepts for efficient query processing. In the following part of the thesis, we will develop specific techniques for efficient time series similarity search under a complex distance measure.
Part IV

Efficient
Time Series Similarity Search
Chapter 8

Dynamic Time Warping (DTW)

In this part of the thesis, we develop novel techniques for efficient query processing of time series similarity search under a complex distance measure, known as Dynamic Time Warping (DTW).

In this chapter, we formalize the DTW and discuss its computation. We also give a quick overview over other complex distance measures for time series. Finally, we present two application scenarios for DTW.

8.1 Introduction to DTW

Time series record the value of one or more attributes as they change over a span of time. For example, temperature measurements from several spatial locations of observation could be recorded at each full hour for each full year enabling year to year climate change analysis.

Formally, a time series is a sequence of (feature) vectors for consecutive points in time:

Definition 8.1 A Time Series $t$ of length $n$ is a temporally ordered sequence $t = [t_1, \ldots, t_n]$, where $t_i$ denotes the value related to a point in time $i$. In case of multivariate data, a point in time $i$ is mapped to a $d$-dimensional attribute vector $t_i = (t_{i1}, \ldots, t_{id})$.

In many applications, time series are very long. For example, stock data and temperature measurements are recorded at high frequency over long
Dynamic Time Warping (DTW)

Figure 8.1: Example for the Euclidean distance and Dynamic Time Warping. The two time series (top blue and bottom red curves) show the same pattern shifted along the time axis; Euclidean Distance (vertical lines) does not capture this, whereas DTW aligns time series and computes the distance between the best match.

periods that may span several years or decades. This observation induces that similarity measures for time series have to scale well with the length of the time series in order to be useful for these applications.

Several different time series similarity models have been proposed, and an extensive survey can be found in [DTS+08]. The most widely used distance function for assessing the similarity between time series is the Euclidean distance, introduced in Chapter 6 (Definition 6.2). Comparing univariate time series based on this distance is straightforward in an element by element manner. The differences between values of corresponding points in time are squared and summed up: $\text{dist}_{L^2}(s, t) := \sqrt{\sum_{i=1}^{n} (s_i - t_i)^2}$.

The Euclidean distance (as well as the other $L_p$-Norm distances) ignores differences in the scale of the time axis as well as shifts along the time axis. As a consequence, two time series that exhibit a very similar pattern from the user’s point of view might incur a high distance value when compared using Euclidean distance. In many domains including climate research, speech-processing, and gene expression analysis, however, time series may be “out of sync”; that is, there are local disturbances on the time axis [AC01, CLC04, KWX+06, AK09]. This is illustrated in Figure 8.1: the time series (top blue curve and bottom red curve) contain the same “peak” pattern, only shifted along the time axis. Euclidean distance (left) compares only values for identical time points, meaning that differences at each point

(a) Euclidean distance
(b) Dynamic Time Warping
in time (thin vertical black lines) are accumulated, and the similar pattern is not captured. This limitation is overcome by one of the most popular distance measures for time series, DTW. It aligns time series by local stretching and shifting along the time axis, i.e. the time is warped. This is illustrated in Figure 8.1 (right), where DTW (thin black lines) matches more similar values, even if they do not occur at the exact same point in time. DTW stems from the speech processing community [BC94] and has since been successfully used in many other domains [AC01, CLC04, KWX+06, AK09].

For long time series, infinite warping is typically not desirable. The result can be a degenerated alignment, as illustrated in Figure 8.2(a): the two dissimilar time series are considered similar according to DTW. To avoid degenerated matching where many values of one time series are matched to very few values of the other one, warping is usually restricted via global constraints termed bands. A band describes how much warping is allowed (i.e., how far apart any two aligned points can be with respect to the time axis). As shown in [DTS+08], bands can enhance both accuracy and efficiency of computation. A illustration for DTW with a band constraint is shown in Figure 8.2. The time series are the same as in Figure 8.1.

DTW is the distance of the best alignment between time series stretched along the time axis, minimizing the overall warping costs. The distance between the aligned points in time is usually measured by one of the $L_p$-norm distances, and the extension of DTW to multivariate time series is straightforward. In this thesis, we apply the Manhattan distance. DTW with a band constraint is defined recursively on the length of the sequences.
**Definition 8.2 k-band DTW.** The Dynamic Time Warping distance between two time series \( s, t \) of length \( n, m \) \((w.l.o.g. \ n \leq m)\) with respect to a bandwidth \( k \) is defined as:

\[
DTW([s_1, \ldots, s_n], [t_1, \ldots, t_m]) = dist_{band}(s_n, t_m) + \min \begin{cases} 
   DTW([s_1, \ldots, s_{n-1}], [t_1, \ldots, t_{m-1}]) \\
   DTW([s_1, \ldots, s_n], [t_1, \ldots, t_{m-1}]) \\
   DTW([s_1, \ldots, s_{n-1}], [t_1, \ldots, t_m])
\end{cases}
\]

with

\[
dist_{band}(s_i, t_j) = \begin{cases} 
   dist(s_i, t_j) & |i - \lceil \frac{jm}{m} \rceil| \leq k \\
   \infty & \text{else}
\end{cases}
\]

\( DTW(\emptyset, \emptyset) = 0, \quad DTW(x, \emptyset) = \infty, \quad DTW(\emptyset, y) = \infty \)

DTW is defined recursively on the minimal cost of possible matches of prefixes shorter by one element. There are three possibilities: match prefixes of both \( s \) and \( t \), match \( s \) with the prefix of \( t \), or match \( t \) with the prefix of \( s \). The difference between overall prefix lengths is restricted to a band of width \( k \) in the time dimension by setting the cost of all overstretched matches to infinity.

The above definition is known as Sakoe-Chiba band where the bandwidth is fixed [SC78]. Another frequently used band, the Itakura band, adjusts the band such that less warping is allowed at the beginning and end [Ita75]. An adaptive approach is presented in [RK04], where band constraints are learned from data. In this thesis, we assume the Sakoe-Chiba band for ease of discussion.

Euclidean distance can be seen as a special case of DTW with bandwidth 0. This corresponds to no warping along the time axis, and thus reduces DTW to summing up the differences of values for corresponding points in time only. In case of possible infinite warping on the time axis \((k = \infty)\), k-band DTW corresponds to DTW without band constraints.

**Computation**

DTW can be computed via dynamic programming in \( O(m \cdot n) \) time, where \( m, n \) are the lengths of the time series. Using a k-band, this is reduced to
8.1. Introduction to DTW

\[ \text{dist}(s_i, t_j) + \min\{c_{i-1,j-1}, c_{i,j-1}, c_{i-1,j}\} \]

Figure 8.3: Warping matrix of DTW with the Sakoe-Chiba band (marked in gray). Only the green path is valid with respect to this band.

\(O(k \times \max\{m, n\})\). Time series of different length can be easily interpolated to uniform length without degrading the quality of the final result in a statistically significant way [RK05]. Instead of computing all possible alignments between the two time series, the recursive definition of DTW is used to fill a cumulative distance matrix. Each matrix entry corresponds to the best alignment between the sub-time series of the corresponding length.

This is illustrated in Figure 8.3. The two time series are depicted to the left (rotated) and at the top of the figure. Their optimal alignment corresponds to matching of the points in time as indicated by the black line in the matrix. For example, the horizontal segment at the lower left indicates that the first six points in time of the time series at the top are matched to the first element of the time series at the left. Assuming a bandwidth of two, the black path is invalid. The green path above it corresponds to the best alignment under this bandwidth constraint.

The DTW cumulative distance matrix is filled analogously to the formula in Definition 8.2. More precisely, a matrix \(C = [c_{i,j}]\) for two time series \(s = [s_1, \ldots, s_n]\) and \(t = [t_1, \ldots, t_m]\) is filled, where each entry \(c_{i,j}\) corresponds to the minimal distance between the subsequences \([s_1, \ldots, s_i]\) and \([t_1, \ldots, t_j]\).
Definition 8.3 **The Cumulative Distance Matrix** $C = [c_{i,j}]$ for two time series $s = [s_1, \ldots, s_n]$ and $t = [t_1, \ldots, t_m]$ is computed recursively such that its entries are calculated as

$$c_{i,j} = \text{dist}(s_i, t_j) + \min\{c_{i-1,j-1}, c_{i-1,j}, c_{i,j-1}\}$$

$$c_{0,0} = 0, \quad c_{i,0} = \infty \text{ for } i \geq 1, \quad c_{0,j} = \infty \text{ for } j \geq 1$$

Thus, the entries of the cumulative distance matrix are filled with the minimum of the three cases to be considered (cf. Figure 8.3). The best alignment between the time series is recursively obtained from the alignments that are possible for the time series shorter by one. Intuitively, this means that the entry $c_{i,j}$ is computed from its three adjacent entries $c_{i-1,j-1}$, $c_{i-1,j}$, and $c_{i,j-1}$ to its bottom left.

The calculation of matrix entries proceeds column-wise. Starting with entry $c_{1,1}$ the entries within the band of each column are calculated before continuing with the next column to the right. The DTW distance is the value of entry $c_{n,m}$ in the top right corner. It corresponds to an alignment called the warping path, which can be visualized in the cumulative distance matrix as a sequence of adjacent entries, beginning in cell $(0,0)$ and ending in cell $(n,m)$. Warping path constraints are expressed by initializing the matrix entries that are not within the band to infinity.

Caused by the possible replication of elements in the alignments, DTW violates both definiteness and triangle inequality [MÖ7] and is therefore not a metric distance function (cf. Definition 6.1 on page 110); accordingly, DTW should not be used with metric indexing structures, because they rely on the triangle inequality to prune the search space and, thus, completeness can no longer be guaranteed. Direct usage in combination with R-Trees is also not a good choice, because a $\text{minDist}$ function for DTW is both inefficient ($O(n^2)$) and ineffective (i.e., a large solution space resulting in a low minimal distance). As a solution, a DTW filter function was introduced that has no dimensional interdependence [Keo02]. It is based on a linearization of the DTW computation, i.e., the DTW is computed component-wise. For this filter, a construction of a $\text{minDist}$ function is straightforward. Several DTW filters are discussed in Chapter 9.
Other Time Series Similarity Models

In this thesis, we concentrate on DTW. In this paragraph, we give an overview over other time series distance measures. In the concept of Longest Common Subsequences (LCSS) [VKG02], dissimilar parts of time series are not used in the similarity estimate. In the Edit Distance on Real Sequences (EDR) [COO05], passages with a specific amount of noise are ignored. The Edit Distance With Real Penalty (ERP) [CN04] is similar to EDR, but ERP is also a metric in contrast to DTW, LCSS, and EDR. SpADe [CNOT07] measures similarity based on matching segments and the distribution of those segments on the time axis. An overview over these models can be found in [CN04]. These applicability of these measures is highly dependent on the application domain, and in [DTS+08, WMD+13] it was shown DTW is on a par with the other similarity models. Our techniques introduced in the following chapters, could, however, be adapted to many of these other models.

8.2 DTW Applications

In Chapters 9 and 10 we introduce novel techniques for efficient processing of DTW queries. In this section, we give two application examples, where we successfully applied the DTW and in which efficient processing is crucial.

*Fast Detection of Suspicious Stream Patterns*

In many application domains, production and information systems have to be routinely monitored. For example, temperature and humidity of a manufacturing laboratory, or incoming and outgoing network traffic in a computing environment are important for surveillance of vital company assets. Successful monitoring should be capable of effectively identifying patterns in streaming data which might indicate critical events. In highly dynamic

---

*An article on this demonstration system has been published in the Proceedings of the 13th International Conference on Extending Database Technology (EDBT/ICDT 2010) [AKGS10].*
systems, and in systems requiring to follow a large number of measurements and several stream sources, efficiency of the detection is crucial. Especially if several patterns for different problematic scenarios are monitored, they have to be compared against the stream in a highly efficient manner.

We proposed an interactive system for monitoring continuous streams of data and for immediate signaling of critical situations. As illustrated in Figure 8.4(left), the system visualizes the incoming information stream(s) and the critical pattern(s) that need to be detected. Users may interactively choose streams to monitor, set up sensitivity thresholds, as well as define and modify critical patterns. The underlying detection algorithm is based on matching patterns using DTW. Fast query processing is achieved by reliably filtering out candidates via a highly efficient multistep filter-and-refine algorithm, called Anticipatory DTW, which is introduced in Chapter 9 of this thesis. Several tools for statistical analysis of detected patterns are provided, as shown in Figure 8.4(right). Users may choose which time periods they would like to compare, and aggregated views on the stream are available at different levels.
8.2. DTW Applications

Figure 8.5: RAVC model: videos are matched according to DTW alignment (in the time domain) of EMD values (in the image domain).

† Video Copy Detection

Video copy detection algorithms automatically detect video content identical to the query or representing an altered version of the original video [HHB02, YST03, LTCJ+07]. The aim is to discover videos that have undergone technical or manual changes, such as change in contrast or editing of the order of scenes in the video [LTCJ+07]. Content-based copy detection is based on the video content alone, i.e. ‘the media itself is the watermark’ [HHB02]. Changes in video content can be roughly categorized in two groups: First, the video may be altered visually in the image domain, as e.g. in the contrast change example. Second, the video may have been reordered in the temporal domain, as e.g. in the frame rate change example.

We proposed a robust adaptable video copy detection scheme (RAVC) that allows effective detection of changes in time and image content. Our technique integrates powerful adaptable distance functions for both visual and temporal alterations. DTW is used for the time domain, and for the visual domain the Earth Mover’s Distance (EMD) is used, which mimics human’s perceived similarity of images [RTG00]. The general idea of RAVC is

†RAVC has been published in the Proceedings of the 11th International Symposium on Spatial and Temporal Databases (SSTD 2009) [AK09]. FastRAVC has been published in the Proceedings of the 15th International Conference on Database Systems for Advanced Applications (DASFAA 2010) [AKS10].
illustrated in Figure 8.5. We evaluated RAVC under different video benchmark scenarios.

Calculation of a matching in both the image and the time domain is a computationally expensive task, because DTW is of quadratic complexity and EMD is of worst case exponential complexity. We proposed FastRAVC, a VA-File-based [WSB98] algorithm for speeding up copy detection under RAVC. For efficient processing, we propose a multistep framework, exploiting properties of the underlying EMD and DTW distance functions. We developed a \textit{Dual minDist} function for the EMD, characterized a class of filters for which we prove completeness of the result, and provide further runtime improvements by a novel tight approximation.

\textbf{Summary}

Summarized, there are many possible application scenarios in which DTW is applicable, and efficient DTW processing is of uttermost importance.

In the next chapters, we develop novel techniques for speeding up DTW queries. In Chapter 9 we introduce a method for accelerating DTW queries in filter-and-refine frameworks. It exploits information unused by existing frameworks with nearly no computational overhead, and it can be flexible combined with existing DTW filter functions and indexing structures. In Chapter 10 we present the new concept of multiple DTW queries, in which new definitions of range and \( k \)NN queries are introduced with respect to multiple queries. Our technique applies filter-and-refine, and in contrast to existing frameworks, multiple different queries are processed simultaneously.
Chapter 9

Anticipating Distances for Efficient DTW Query Processing

* DTW is a high-quality distance measure and is used in many application domains. As it is computationally expensive, query processing algorithms are needed that minimize the number of DTW computations.

In this chapter, we contribute a novel filter-and-refine DTW algorithm called Anticipatory DTW (aDTW). Existing algorithms aim at efficiently finding similar time series by filtering the database and computing the DTW in the subsequent refinement step. Unlike these algorithms, our approach exploits previously unused information from the filter step during the refinement, allowing for faster rejection of false candidates. We characterize a class of applicable filters for our approach, which comprises state-of-the-art lower bounds of DTW. Our novel anticipatory pruning incurs hardly any overhead and no false dismissals. We demonstrate substantial efficiency improvements in experiments on synthetic and real world time series and show that our technique is highly scalable to multivariate, long time series and wide DTW bands.

*This chapter has been published in the Proceedings of the 35th International Conference on Very Large Databases (VLDB 2009) [AWK’09].
9.1 Motivation

Time series data is important for commerce, science, and engineering. In these domains large amounts of time-dependent data are created, acquired, and analyzed. Time series databases are often large, and the contained time series can be long and multivariate. Due to its effectiveness, DTW is the method of choice in many similarity search applications. In its general formulation, DTW is typically too costly for comparing longer time series as the computational complexity is quadratic in the length of time series. Accordingly, filter distances were introduced to integrate DTW into traditional filter-and-refine frameworks, in which the exact DTW distance is computed if the filter distance does not exceed the current pruning threshold. An improvement of this technique is early stopping [SYF05], where the computation is stopped as soon as an intermediate result exceeds the pruning threshold.

However, all the existing filter-and-refine frameworks discard relevant information acquired in the filter step, and recompute this information in the refinement step. This recalculation of already existing information corresponds to a massive computational overhead.

In this chapter, we present a novel approach called Anticipatory DTW (aDTW) for speeding up DTW-based retrieval on both univariate and multivariate time series. Intuitively speaking, aDTW anticipates the DTW refinement result by exploiting previously computed information in multistep filter-and-refine algorithms.

Our approach goes far beyond early stopping techniques. In each step of a DTW computation, we derive a lower-bounding anticipatory distance that takes the intermediate DTW distance computed so far and greatly improves the approximation quality of the lower bound by anticipating the remaining steps of the exact DTW computation. This anticipatory component is derived from the previously computed filter step, thus re-using information that is ignored in existing approaches. In this manner, the incremental DTW computation is stopped as soon as the anticipatory distance exceeds the pruning threshold. We prove that aDTW is lossless for the most widely used DTW lower bounding filters. As aDTW is orthogonal to existing indexing and di-
9.2. Comparison with Related Work

Techniques for fast processing of DTW queries can be categorized into approximate or exact with respect to the result set they deliver.

A number of approximate techniques have been proposed for speeding up DTW queries. [YJF98] proposes an approximate embedding of DTW into Euclidean space. This technique is extended to a Haar wavelet transform based technique for fewer false positives [CFY03], but possibly more false negatives. Iterative Deepening DTW computes different levels of dimensionality reduction from piecewise linear approximations [CKHP02]. Using a probabilistic model based on sample approximation errors, time series are either pruned or compared at a finer level of approximation. FastDTW computes approximations of the warping path at different levels of granularity [SC04].

Embedding-Based Subsequence Matching hashes subsequences of the original time series to vectors based on their DTW distance to reference time series [APP+08]. Subsequences that are identified as potentially similar to the query are then refined using the DTW. All of these techniques provide efficiency gains by sacrificing the correctness of the result as they are approximate in nature. Our approach guarantees correct results.

Correctness of the result is guaranteed in lower bounding filter-and-refine techniques [Fal96, SK98]. The Piecewise Aggregate Approximation (PAA) representation is composed of averages of consecutive values along the time axis [KCPM01]. The Adaptive Piecewise Constant Approximation adapts the number of values that are averaged to reduce the approximation error [KCPM00]. For these representations, lower-bounding filter functions for
DTW have been proposed. Another lossless approach is based on four characteristic values of time series: their starting and ending value, their minimum, and their maximum [KPC01]. $LB_{Keogh}$ provides a lower-bound that uses the warping path constraint to compute an envelope on the warping cost for PAA segments [Keo02]. This approach has been extended to 2-dimensional envelopes around trajectories in [VHGK03, RM02]. Improved versions of the envelope technique have been proposed in [ZS03] and [ZW07]. The Fast search method for Time Warping (FTW) uses different resolutions to gradually refine the optimal warping path [SYF05]. Using early stopping, only the warping paths with a DTW distance not exceeding some pruning distance are evaluated. This technique is also known as early abandoning [KWX+06].

Our technique can be integrated with these lower bounding filter functions to improve performance. We give more details in Section 9.3 and discuss their possible integration in Section 9.3.7.

### 9.3 Anticipatory DTW

Our approach extends the classical algorithms for DTW that use a multistep filter-and-refine architecture. We term our technique Anticipatory DTW, or short aDTW. It is based on a number of properties (Sections 9.3.3 to 9.3.5) and makes use of our anticipatory pruning distance (Section 9.3.6). We prove that the anticipatory pruning distance is a lower bound to DTW implying that aDTW guarantees no false dismissals (Theorem 9.3). We start with an overview over aDTW and preliminaries in Sections 9.3.1 and 9.3.2.

#### 9.3.1 Overview Over Our Proposed Method

We propose a technique that is orthogonal to existing DTW speed-up techniques based on filter-and-refine frameworks. It plugs into these frameworks as illustrated in the right part of Figure 9.1. Instead of directly computing the complete (and thus costly) DTW refinement after the filter step, our Anticipatory Pruning Distance (APD) is incrementally computed. If it exceeds the given pruning threshold $max$ at any time, the time series can be ruled out as
9.3. Anticipatory DTW

Figure 9.1: Anticipatory pruning in filter-and-refine algorithms.

part of the result set – safely discarding many computation steps. If no pruning is possible in intermediate steps, the last step produces the exact DTW distance. By using distance information from the filter step for the APD in the refinement step, we are able to provide a lower bounding estimate of the still to be calculated DTW steps. Thus, we go beyond previous approaches known as early stopping [SYF05] or early abandoning [KWX+06], which also partially avoid the computation of DTW distances but do not anticipate future computation steps.

Our APD is based on three properties: First, DTW computation is incremental, i.e., the entries in the warping matrix increase with the sequence length. Second, many existing lower bounding filters for DTW can be characterized as piecewise, i.e., a subsequence of the filter computation is a valid filter for the subsequences themselves. And finally, DTW is reversible, i.e., computing the warping path from the beginning to the end is the same as doing so from the end to the beginning.

The aDTW algorithm uses these three properties by calculating piecewise filter distances on reversed time series in the filter step of a filter-and-refine framework. During the incremental computation of the DTW, the information from the piecewise filter is a lower bounding anticipatory estimate of the DTW parts yet to be calculated. The exploitation of this additional information allows for stopping the DTW calculation much sooner than it would have been possible without the anticipatory part. If no stopping is possible, the last step of computing APD corresponds to the last step of computing the DTW, i.e. there is no independent refinement step.
Our approach is highly flexible because it can be combined with existing lower bounding filters. Integration with these is easy, as all information required for anticipatory pruning is derived from existing filter calculations. Moreover, the pruning capability of aDTW comes at hardly any additional cost, as the filter distances have been already computed in the filter step.

9.3.2 Preliminaries

As some of the theorems and proofs in this chapter depend on the warping path within a given valid band for DTW (cf. Section 8.1), the valid matrix band and the warping path are defined in the following.

Definition 9.1 Valid Matrix Band. The valid cells in the cumulative k-band DTW distance matrix of size $n \times m$ (w.l.o.g. $n \leq m$) with respect to a band parameter $k$ are

$$
\text{band} = \bigcup_{1 \leq j \leq m} \text{band}_j
$$

with $\text{band}_j$ as the valid cells in column $j$:

$$
\text{band}_j = \{(i,j) \mid 1 \leq i \leq n, \left| i - \left\lceil \frac{j \cdot n}{m} \right\rceil \right| \leq k\}
$$

We assume the Sakoe-Chiba band, but the definition can be easily adapted to other band types as well.

Definition 9.2 The Warping Path $P$ for two time series $s = [s_1, \ldots, s_n]$ and $t = [t_1, \ldots, t_m]$ is defined as a series of matrix cells

$$
P = p_1, \ldots, p_l \text{ with } p_1 = (0,0), \ p_l = (n,m)
$$

For any two $p_i = (k,l), p_{i+1} = (q,r)$ of a warping path $P$ in a cumulative distance matrix the following holds:

- **monotony:** $q - k \geq 0 \land r - l \geq 0$
- **continuity:** $q - k \leq 1 \land r - l \leq 1$
- **alignment:** $c_{q,r} = \text{dist}(s_q, t_r) + c_{k,l}$
A warping path is monotonic in the sense that the time does not go backwards. The path is continuous in that there are no omitted points in time; successive path elements correspond to consecutive points in time. The alignment property ensures that each cell $p_i$ arose from its direct predecessor $p_{i-1}$.

In the next three sections, the properties important for aDTW are discussed.

### 9.3.3 Incremental Computation of DTW

Anticipatory pruning uses filter information to perform additional pruning checks in the refinement. These checks are based on our new anticipatory estimate and the fact that DTW is incremental (i.e., the column minima of the cumulative distance matrix increase with the length of the time series).

**Theorem 9.1** DTW is incremental. For any cumulative distance matrix $C = [c_{i,j}]$ as defined in Definition 8.3, the column minima are monotonically non-decreasing

$$\min_{i=1,...,n}\{c_{i,x}\} \leq \min_{i=1,...,n}\{c_{i,y}\} \text{ for } x < y.$$ 

**Proof 9.1** The proof is straightforward from the definition of the cumulative distance matrix: $c_{i,j} = \text{dist}(s_i, t_j) + \min\{c_{i-1,j-1} \; , \; c_{i,j-1} \; , \; c_{i-1,j}\}$. Any element $c_{i,y}$ in a column $y$ is based on at least one value from the same or a preceding column plus a non-negative distance between the two time series entries $s_i$ and $t_y$. If the value is from the preceding column, the theorem holds. If the value is from the same column, it is based on the entry immediately below (i.e., $c_{i-1,y}$). For this entry, the same argument holds. There is a finite number of steps in which the entry can be based on an entry in the same column. As soon as the boundary of the $k$ band is reached, the choice of minimum has to be based on the preceding column. Thus, the theorem holds.

Consequently, many DTW computations do not have to be fully processed, but can be interrupted once the column minimum exceeds the $\max$ value. A similar idea, called early stopping, in the FTW approach is used to avoid refinement of approximate warping paths at different levels of granularity.
In our anticipatory pruning technique, we use Theorem 9.1 for exact DTW computations, but go beyond column minima as we add pieces of filter information for a much closer estimate of the exact DTW distance.

Entries $c_{i,j}$ in the cumulative distance matrix correspond to the cost of possible warping paths between subsequences $[s_1, ..., s_i]$ and $[t_1, ..., t_j]$. For the remaining alignment and its cost, we do not yet have the entries in the cumulative distance matrix. We do, however, have a filter distance for the warping path of the complete time series. Provided that we can decompose this filter into one that serves as a lower bound on the remaining warping path, we may combine the filter components with the column minimum for an overall estimate which we call Anticipatory Pruning Distance.

### 9.3.4 Piecewise Filter Computation for DTW

Anticipatory pruning can be understood as an incremental refinement of the filter with DTW distance information. As illustrated in Figure 9.2, the cumulative distance matrix for DTW computation is filled as usual. For example, columns 1 to 5 have been computed in the leftmost figure. As seen above, the column minimum in the fifth column is a lower bound for the DTW distance between the two time series depicted to the left and at the top of the cumulative distance matrix. We do know, however, that only partial time series have been accounted for, namely up to the fifth entry in the time series at the top, and up to the seventh entry (5 plus band of width 2) in the time series to the left. For the remaining subsequences of the time series, starting
in the sixth column, we derive an estimate from the preceding filter step. This is illustrated in the darker area at the top right for columns 6 to 15. The estimate is a piece of filter information corresponding to these last columns (the anticipatory part).

This estimate on partial time series alignments requires the filter to be piecewise, i.e., decomposable into a series of lower bounds for all subsequences of increasing lengths that start at the beginning of the time series.

**Definition 9.3 Piecewise DTW lower bound.** A piecewise lower bounding filter for the DTW distance is a set \( f = \{ f_0, \ldots, f_m \} \) with the following property:

\[
\begin{align*}
  j = 0 : & \quad f_j(s,t) = 0 \\
  \forall j > 0 : & \quad f_j(s,t) \leq \min_{(i,j) \in \text{band}_j} \text{DTW}([s_1, \ldots, s_i], [t_1, \ldots, t_j])
\end{align*}
\]

Intuitively speaking, a piecewise lower bounding filter can be decomposed into a series of lower bounds for all possible partial warping paths that start at the beginning of both time series and end in a respective column. The piecewise property is not a major constraint, as most existing lower bounding filter for DTW can be decomposed in such a manner. We will prove this property for some of the most widely used approaches in Section 9.3.7.

**9.3.5 Reversible Computation of DTW**

The piecewise property of a DTW lower bound and the incremental computation property of DTW together do not suffice to be able to combine pieces of the filter with partial DTW computation for a close lower bound of DTW. To derive an overall lower bound, we require a combination of a partial DTW computations up to some column \( j \) of the cumulative distance matrix and piecewise filter information for the remaining columns \( (j + 1), \ldots, m \). However, the piecewise filter only provides information for ranges of columns starting with the first column. Fortunately, DTW computation is reversible, i.e., computing the distance for the reversed time series (from the end to the beginning) yields exactly the same result as computing it for the original time series.
**Theorem 9.2** DTW is reversible. For any two time series \([s_1, ..., s_n]\) and \([t_1, ..., t_m]\), their DTW distance is the same as for the reversed time series:

\[
\text{DTW}([s_1, ..., s_n], [t_1, ..., t_m]) = \text{DTW}([s_n, ..., s_1], [t_m, ..., t_1])
\]

**Proof 9.2** Assume that this does not hold and to the contrary (w.l.o.g.):

\[
\text{DTW}([s_1, ..., s_n], [t_1, ..., t_m]) < \text{DTW}([s_n, ..., s_1], [t_m, ..., t_1]).
\]

Let \(p_1, ..., p_l\) and \(p_1^-, ..., p_l^-\) be warping paths with minimal cost for the non-reversed and for the reversed time series. Since reversing path \(p_1, ..., p_l\) also yields a valid warping path for the reversed time series (properties monotony, continuity, and alignment of Definition 9.2 follow from the according properties of the non-reversed path) and since its cost equals \(\text{DTW}([s_1, ..., s_n], [t_1, ..., t_m])\), it follows that \(p_1^-, ..., p_l^-\) cannot be a warping path with minimal cost. As this contradicts the assumption, DTW is reversible.

Reversibility is important for our anticipatory pruning, as it allows deriving a lossless estimate of the DTW from any piecewise lower bounding filter. By reversing the order of time series during filter computation, we obtain the same filter distance. However, we may now use the pieces of the filter computation from the end of the time series backwards up to the current point of DTW computation as an estimate of the remaining path cost. As this estimate is lower bounding, and so are column minima, anticipatory pruning is lower bounding as well, as we will prove formally in Theorem 9.3.

### 9.3.6 Anticipatory Pruning Distance

Our aDTW includes a sequence of pruning checks. After each computation of a column in the cumulative distance matrix, piecewise filter information is added to the information obtained from that column, yielding an overall lower bound for the exact DTW distance. It therefore corresponds to a series of \(m\) filters for time series \(s, t\) of length \(n, m\), where the final step provides the actual DTW distance. Each of the individual filter steps has a computational complexity that is linear with respect to the DTW bandwidth \(k\), i.e. \(O(k)\).

Formally, our novel Anticipatory Pruning Distance is defined as:
Definition 9.4 Anticipatory Pruning Distance (APD). Given two time series \( s \) and \( t \) of length \( n \) and \( m \), a cumulative distance matrix \( C = [c_{i,j}] \) according to Def. 8.3, a piecewise lower bounding filter \( f \) for the reversed time series \( s^\rightarrow \) and \( t^\rightarrow \) according to Def. 9.3, and a value \( j \in \{1, \ldots, m\} \), the \( j^{th} \) step of the APD for the \( k \)-band DTW distance between \( s \) and \( t \) is defined as

\[
APD_j(s, t) := \min_{i=1,\ldots,n} \{c_{i,j}\} + f_{m-j}(s^\rightarrow, t^\rightarrow).
\]

The definition of anticipatory pruning distance is thus a set of DTW column minima with added pieces of filter information for the remainder of the time series. During each incremental computation step \( j \), the column minimum of the partial path between \([s_1, \ldots, s_i]\) and \([t_1, \ldots, t_j]\) according to DTW is combined with the anticipatory information for the rest of the alignment available from the lower bound from the filter step. The resulting estimate of the entire path in step \( j \) is \( APD_j \). W.l.o.g. and for sake of simplicity, we assume a DTW bandwidth of \( k = \infty \): the minima are computed over the whole extent of the columns (\(\min_{i=1,\ldots,n} \{c_{i,j}\} \) for filter step \( j \)). This is a worst case assumption, since usually the minima is obtained from a \( k \)-band much smaller than the time series length (\(\min_{i=j-k,\ldots,j+k} \{c_{i,j}\} \)).

Anticipatory pruning is illustrated in the example in Figure 9.3. The upper part of the figure shows two univariate time series: a query \( q \), a database element \( t \), and the corresponding cumulative distance matrix. For reasons of simplicity, this example uses an \( L_1 \)-based variant of the \( LB_{Keogh} \) lower bound (which is piecewise as discussed later on). The relationship between the DTW distance (35), the pruning threshold (22), and the components of \( APD_j \) are depicted in the lower part of Figure 9.3.

The filter distance between \( q^\rightarrow \) and \( t^\rightarrow \) yields the following piecewise results: \( f(q^\rightarrow, t^\rightarrow) = (0, 7, \ldots, 18) \). Thus, the filter step distance is 18. To make the computation of our APD more intuitive to the reader, we reversed the piecewise results in the figure: \( f^{-}_j(s^\rightarrow, t^\rightarrow) := f_{m-j}(s^\rightarrow, t^\rightarrow) \). Assume that our current pruning threshold \( max \) is 22. Hence, the filter step distance of 18 does not exceed \( max \), and \( t \) cannot be pruned by the filter. As illustrated at the top of Figure 9.3, we start filling the cumulative distance matrix for
Figure 9.3: Anticipatory pruning example for two univariate time series. Anticipatory pruning takes place at time step $j = 5$. Early stopping would have needed all 10 time steps.

$DTW(q, t)$ and compute the minimum of column $j$ (denoted by $\text{min}()$) in step $j$. For example, the first column minimum is 2, which accounts for the mappings of the 1-prefix of $t$ to those prefixes of $q$ permitted by $\text{band}_1$. As can be seen in the figure, the minima of the first few columns are only loose bounds. For anticipatory pruning, we add the first filter entry $f_1^-$, which lower bounds all possible reversed alignments starting in (10,10) and ending in column 2. We obtain “$\text{min} + f" = 2 + 18 = 20$ as first APD, which is much closer to the DTW distance than the column minimum alone. Yet it still is less than the pruning threshold $\text{max}$. Continuing with matrix filling, the second column minimum is 5, and the filter entry $f_2^-$ is 15, yielding 20 again. We proceed until step 5, where the column minimum is 13. Combined with the corresponding filter distance of 10, the sum is 23, exceeding the pruning...
threshold max. We immediately stop the DTW computation, skip the remaining five columns, and discard t. In this example, early stopping without our anticipatory component takes 5 more steps before the time series can be pruned; accordingly, our technique saves considerable computations in the cumulative distance matrix of DTW at the costs of one extra addition per matrix column.

We now prove that the APD is indeed a lower bound of the DTW and therefore guarantees lossless query processing:

**Theorem 9.3 APD lower bounds DTW.** The anticipatory pruning distance \( APD_j \) according to Def. 9.4 lower bounds the \( k \)-band DTW distance between two time series \( s \) and \( t \) of length \( n \) and \( m \):

\[
APD_j(s, t) \leq DTW(s, t) \quad \forall j \in \{1, ..., m\}
\]

**Proof 9.3** Anticipatory pruning is essentially a series of partial DTW paths combined with a lower bound estimate of the remainder taken from the previous filter step. We show that it lower bounds DTW by arguing that (1) for all possible steps \( j \), \( 1 \leq j \leq m \), the column minimum lower bounds the true path. It holds (2) that DTW is reversible, i.e., we can replace a path between two time series by the path between reversed time series. Finally, in (3) we show that we can combine the first two properties to derive anticipatory pruning as a lower bound of DTW.

(1) The cost of any DTW warping path ending in column \( j \) of the cumulative distance matrix \( C = [c_{i,j}] \) is obviously lower bounded by the minimum cost of all these paths \( \min_{i=1,...,n} \{c_{i,j}\} \).

(2) As Theorem 9.2 states, DTW is reversible. Thus, we can make use of the fact that \( DTW([s_1, ..., s_n], [t_1, ..., t_m]) = DTW([s_n, ..., s_1], [t_m, ..., t_1]) \) in (3).

(3) We now prove that anticipatory pruning, as the sum of a column minimum and a lower bound of any valid path that starts in the subsequent column (or reverse path that ends in this column), lower bounds DTW.

\[
APD_j(s, t) := \min_{i=1,...,n} \{c_{i,j}\} + f_{m-j}(s^-, t^-) \leq DTW(s, t)
\]
Due to the continuity of DTW, the minimal path (see Definition 9.2) passes through all columns, and thus also through both column \( j \) and column \( j + 1 \). The optimal path \( P = p_1, \ldots, p_l \) for \( \text{DTW}(s, t) \) can be decomposed into two parts: the first part from the beginning to column \( j \): \( p_1, \ldots, p_u \) and one which continues in column \( j + 1 \) to the end: \( p_u+1, \ldots, p_l \). (If more than one of the \( p_k \) is in column \( j \), let \( p_u \) denote the last one). Let the positions \( p_u := (x, j) \) and \( p_{u+1} := (y, j + 1) \). Then, we can rewrite the right side of the inequation as \( \text{DTW}(s, t) = c_{p_u} + \text{DTW}([s_y, \ldots, s_n], [t_{j+1}, \ldots, t_m]) \).

From (1), we have that \( \min_{i=1,\ldots,n}\{c_{i,j}\} \leq c_{p_u} \). From (2), we have that \( \text{DTW}([s_y, \ldots, s_n], [t_{j+1}, \ldots, t_m]) = \text{DTW}([s_n, \ldots, s_y], [t_m, \ldots, t_{j+1}]) \), which is underestimated by \( f_{m-j}(s^\rightarrow, t^\rightarrow) \) by definition of \( f \). Thus, we indeed have that anticipatory pruning lower bounds the DTW.

From the proof that anticipatory pruning is a lower bound of the actual DTW, it immediately follows that query processing with anticipatory pruning is lossless in multistep filter-and-refine algorithms. More details and proofs that lower bounding guarantees completeness can be found in [Fal96, SK98]. Moreover, we can easily see from the definition of anticipatory pruning that eventually the series of computations results in the exact DTW distance.

Our approach incurs very little overhead, as we only maintain piecewise information on the filter distances. The computations of the individual filter steps have the same asymptotic complexity as the existing early stopping approaches [SYF05, KWX’06], which is linear (\( O(k) \)) with respect to the DTW bandwidth \( k \). Concretely, for the \( j \)th filter step, \( \min_{i=j-k,\ldots,j+k}\{c_{i,j}\} \in O(k) \) and \( \min_{i=j-k,\ldots,j+k}\{c_{i,j}\} + f_{m-j}(s^\rightarrow, t^\rightarrow) \in O(k) \), since \( f_{m-j}(s^\rightarrow, t^\rightarrow) \in O(1) \). In the filter step, early stopping approaches have a space complexity of \( O(1) \) per time series, while aDTW’s space complexity is \( O(n) \) per time series, caused by the need to store the individual pieces of filter information.

Our approach can easily be generalized to multiple filter steps. As long as filters are both lower bounding and piecewise, anticipatory pruning can be applied to any step in the filter chain. As the filter is computed for increasing time series length, an estimate on the remainder of the time series can be obtained from the previous filter step. To do so, merely the order in which the filter steps process the time series has to be alternated between steps.
Anticipatory pruning is computed as outlined in the pseudo code in Algorithm 9.1. Given two time series \( q, t \), a pruning threshold \( \text{max} \), and a vector \( f = (f_0(q^-, t^-), ..., f_m(q^-, t^-)) \) that represents the distances of the piecewise lower bounding filters, we start by initializing a column vector \( \text{col} \). Then, in each incremental step \( j \), the next column of the exact DTW computation is calculated by \( \text{CalcDTWMatrixColumnBand} \) and the new \( \text{APD}_j(q, t) \) is computed as \( \text{APD}_j \). The value \( \text{APD}_j \) is used for pruning iff it exceeds \( \text{max} \). Otherwise, the next pruning value \( \text{AP}\text{PD}_{j+1} \) is computed. If no pruning is possible, the exact \( \text{DTW}(q, t) \) is returned unless it itself exceeds \( \text{max} \).

In the next section, we study different existing lower bounding filter techniques for DTW. We show that they are indeed piecewise and thus can be used for aDTW.

\[\text{Algorithm 9.1: Anticipatory pruning}\]

1. \textbf{Procedure} \( \text{AP} \) (Time series \( q, t \), Real \( \text{max} \), Vector \( f \))
2. Vector \( \text{col}[q.\text{length} + 1] = [0, \infty, ..., \infty] \) \( \triangleright \) first index is 0
3. \textbf{for} \( j = 1 \) to \( t.\text{length} \) \textbf{do}
4. \( \text{col} = \text{CalcDTWMatrixColumnBand}(q, t, \text{col}, j) \)
5. Real \( \text{APD}_j = \min(\text{col}[1], ..., \text{col}[q.\text{length}]) + f[t.\text{length}-j] \)
6. \textbf{if} \( \text{APD}_j > \text{max} \) \textbf{then}
7. \( \text{return} \infty \) \( \triangleright \) pruning
8. \textbf{if} \( \text{col}[t.\text{length}] > \text{max} \) \textbf{then}
9. \( \text{return} \infty \) \( \triangleright \) pruning
10. \textbf{return} \( \text{col}[t.\text{length}] \) \( \triangleright \) no pruning, return DTW

### 9.3.7 Piecewise Lower Bounding Filters

In this section, we describe state-of-the-art methods providing lower bounding filter distances for DTW. While the methods themselves have been introduced elsewhere, our contribution lies in showing that all of these different methods fulfill the requirement of being piecewise according to Definition 9.3. From this, it follows by Theorem 9.3 that anticipatory pruning of DTW with these methods is lossless.
The lower bounds presented in Sections 9.3.7 and 9.3.7 are only capable of comparing sequences of the same length \( n \), while FTW in Section 9.3.7 can cope with different lengths.

**Linearization**

The basic idea for linearization of the DTW computation for efficient and exact indexing is based on computing an *envelope* of upper and lower values \( U \) and \( L \) around the query time series \( q \) with respect to the \( k \)-band:

\[
U_i = \max_{i-k \leq j \leq i+k} \{ q_j \} \quad \text{and} \quad L_i = \min_{i-k \leq j \leq i+k} \{ q_j \}.
\]

The squared Euclidean distance between values above or below the envelope of the other time series \( t \) lower bound the exact \( k \)-DTW distance [Keo02]. This corresponds to the minimal distance to a MBR in R-Trees.

\[
LB_{Keogh}(q,t) = \sqrt{\sum_{i=1}^{n} \minDist(t_i, L_i, U_i)}
\]

with

\[
\minDist(t_i, L_i, U_i) = \begin{cases} 
(t_i - U_i)^2 & t_i > U_i \\
(t_i - L_i)^2 & t_i < L_i \\
0 & \text{else}
\end{cases}
\]

An extension providing a tighter envelope when PAA dimensionality reduction is applied is given in [ZS03] as:

\[
\overline{U}_i = \frac{N}{n} (u_{n(i-1)+1} + \ldots + u_{n(i)}) \quad \text{and} \quad \overline{L}_i = \frac{N}{n} (l_{n(i-1)+1} + \ldots + l_{n(i)}).
\]

The \( LB_{Keogh} \) approach is a dimension-wise summation of distances. Thus, it constitutes a piecewise lower bound of DTW.

**Theorem 9.4** \( LB_{Keogh} \) is a piecewise lower bounding filter according to Definition 9.3.

**Proof 9.4** Decomposition into a set of filters \( f_j \) for increasing sequence length is straightforward:

\[
f_j = \sqrt{\sum_{i=1}^{j} \minDist(t_i, L_i, U_i)}.
\]
That is, the summation up to the current subsequence length \( j \). For both \( U, L \) and \( \bar{U}, \bar{L} \), [Keo02, ZS03] prove that the lower bounding property holds.

In our experiments, we evaluate anticipatory pruning for \( LB_{Keogh} \) with the tighter envelope \( \bar{U}, \bar{L} \).

### Corner Boundaries

A different type of lower bound can be obtained as piecewise corner-like shapes in the warping matrix through which every warping path has to pass [ZW07]. Formally, for a time series \( q \), its corner shapes are:

\[
C^L_i(q) = \begin{cases} 
\min_{\max(1,i-k) \leq j \leq i} \{q_j\} & i \leq \left\lfloor \frac{n}{2} \right\rfloor, \\
\min_{i \leq j \leq \min(n,i+k)} \{q_j\} & i > \left\lfloor \frac{n}{2} \right\rfloor,
\end{cases}
\]

\[
C^U_i(q) = \begin{cases} 
\max_{\max(1,i-k) \leq j \leq i} \{q_j\} & i \leq \left\lfloor \frac{n}{2} \right\rfloor, \\
\max_{i \leq j \leq \min(n,i+k)} \{q_j\} & i > \left\lfloor \frac{n}{2} \right\rfloor,
\end{cases}
\]

\[C_i(q) = (C^L_i(q), C^U_i(q)).\]

These approximations are constructed for both time series and enclose the positions in the time series that correspond to the corner shapes in the warping matrix. A corner shape \( i \) in the warping matrix for two time series \( t \) and \( q \) is a combination of the two envelope positions \( C_i(q) \) and \( C_i(t) \). In [ZW07], experiments showed that a hybrid approach of corner shapes at the beginning and the end of the warping matrix and straight line shapes in the middle leads to an even tighter lower bound called \( LB_{Hybrid} \). The idea of straight line shapes is very similar to the \( LB_{Keogh} \) approach, thus the hybrid corner method is defined as

\[
LB^2_{Hybrid}(q,t) = \sum_{i=1}^{n} \begin{cases} 
\min\text{Dist}(t_i, L_i, U_i) & k + 2 \leq i \leq n - k - 1, \\
\min\{CD_i(q,t), CD_i(t,q)\} & \text{else},
\end{cases}
\]

with the Corner Distance \( CD_i \) that calculates the minimal distance for a warping path that passes through the corner shape \( i \):
\[ CD_i(q, t) = \begin{cases} 
(q_i - C^U_i(t))^2 & q_i > C^U_i(t) \\
(q_i - C^L_i(t))^2 & q_i < C^L_i(t) \\
0 & \text{else}
\end{cases} \]

**Theorem 9.5** \( LB_{Hybrid} \) is a piecewise lower bounding filter according to Definition 9.3.

**Proof 9.5** Since \( LB_{Hybrid} \) uses the linearization of \( LB_{Keogh} \) in the first case, this part of the proof follows from Theorem 9.4. In the corner case, we immediately obtain the pieces \( f_j \) from the definition of the corner shapes.

In our experiments, we demonstrate the usefulness of anticipatory pruning for corner boundaries in terms of its pruning capability.

**Path Approximation**

Another approach for speeding up DTW is taken in the FTW (Fast search method for Dynamic Time Warping) technique [SYF05]. The idea is to generate approximations of the optimal warping. It works for DTW with and without band constraints by checking continuously for any approximate segment of the time series whether the \( \max \) value of the current candidates has been exceeded. If this is not the case, a finer approximation is generated. The efficiency gains are based on the fact that coarse DTW computations are computationally less expensive.

At differing levels of granularity, segments are approximated using minimum and maximum values \( U, L \). The Lower Bounding distance measure with Segmentation (LBS) is exactly the DTW on the approximated segments. For time series segments of different lengths a normalization factor is used. The distance is refined locally (i.e., each segment is only refined while below the \( \max \) value).

Since LBS is the DTW distance on segments, it also constitutes a piecewise lower bound:

**Theorem 9.6** \( FTW \) is a piecewise lower bounding filter w.r.t. Definition 9.3.
9.3. Anticipatory DTW

Proof 9.6 Let $f_j = DTW([Q_1, ..., Q_J], [T_1, ..., T_{J+K}])$ be a series of piecewise filters, where $Q$ and $T$ are approximations of time series $q$ and $t$. $J$ denotes the smallest value such that the approximated segment $[Q_1, ..., Q_J]$ fully contains all points of the time series up to index $j$. Similarly, $J + K$ denotes the smallest value such that the approximated segment $[T_1, ..., T_{J+K}]$ fully contains all points of the time series up to index $j + k$. While the exact value of $J$ and $J + K$ may vary for individual time series, at least the coarsest level of approximation is computed for the entire time series, yielding valid pieces for the entire length. As the computation is in the same cumulative matrix procedure as for DTW, we obtain a piecewise filter via column minima as desired. For the lower bounding property see [SYF05].

We study the performance of anticipatory pruning for FTW in our experimental section.

Dimensionality Reduction and Indexing

Dimensionality reduction is a very useful technique for efficiency gains in time series similarity search as many time series are very long, i.e., high dimensional. Several approaches have been suggested, such as the Piecewise Aggregate Approximation (PAA) used, e.g., in the $LB_{Keogh}$ approach. The idea is to replace parts of the original time series by constant approximations. All segments, either of fixed or of adaptive length (e.g., Adaptive Piecewise Constant Approximation (APCA) [KCPM00]), are piecewise by their very na-
Figure 9.5: Absolute efficiency improvement (average query time) for different reductions on RW2. Average query times are given on a log scale.

ture. Consequently, they fulfill the requirements for anticipatory pruning and can be used with our aDTW as well.

A number of DTW speed-up techniques also use indexing structures (e.g., R-trees in [Keo02] or sequential structures in [SYF05]). As anticipatory pruning is orthogonal to such techniques, the efficiency benefit is maintained in our approach. It requires merely a change in the computation of the refinement step, which is independent of any underlying indexing approach.

9.4 Experiments

All experiments were executed on 2.33GHz Intel XEON CPU cores running JAVA implementations. The following default parameters were used where not stated otherwise: width of DTW band \( k = 40 \), length of the time series \( n = 512 \), number of nearest neighbors retrieved 10 (per query). The query workload was 200. Several synthetic datasets for scalability studies and real world datasets for different parameters were used. For scalability in the number of attributes per point in time, we generated two multivariate random walk datasets (univariate examples are shown in Figure 9.4). Both contain time series of length 512 and are of cardinality 10,000. The number of attributes \( d \) was varied between 1 and 50.

**RW1:** The non-normalized value of the \( j^{th} \) component \( (j \in \{1,...,d\}) \) of \( t_i+1 \) is a random value drawn from a normal distribution with parameters \( \mu = 0, \sigma^2 = 1 \) added to the value of the \( j^{th} \) component of \( t_i \): \( t_{i+1}, j = t_{i, j} + N(0,1) \). RW1 was normalized to an average value of 0.
RW2: The first two elements of the time series are generated as in RW1. For the remaining points in time of RW2, the average value $\mu$ depends on the last increase/decrease: $t_{(i+1)j} = t_{ij} + N(t_{ij} - t_{(i-1)j}, 1)$. As early values of RW2 have a low variance while the later values have a high variance, RW2 was normalized to an average value of 0.

In addition to these synthetic time series, we use three multivariate real world datasets.

**SignLanguage:** This multivariate dataset is derived from the 11 real valued attributes of the sign language finger tracking data from [Kad99]. For the efficiency experiments, time series were created from the concatenated raw data by extracting non-overlapping windows of equal length. The length $n$ of the non-overlapping time series was between 64 and 512 and the band $k$ was between 10 and 150. The number of time series was fixed at 1,400.

**TRECvid & NEWSVid:** We use two video datasets. The first one is the TRECvid benchmark data [SOK06]. The second dataset, NEWSVid, consists of TV news we recorded at 30 fps. 20-dimensional HSV histograms were computed for each video frame (i.e., the time series are multivariate with $d = 20$). The length $n$ of the non-overlapping video sequences was between 64 and 2048 frames. The cardinality of the database depends on the length of the time series and varies between 650 and 2,000 for the TRECvid data and between 2,000 and 8,000 for the TV news.

In the experiments, we thoroughly investigate the runtime improvements of anticipatory pruning over the piecewise base techniques Linearization ($LB_{Keogh}$), Corner boundaries ($LB_{Hybrid}$), and Path Approximation ($FTW$).
9.4.1 Dimensionality Reduction

We start by evaluating the efficiency of anticipatory pruning with respect to dimensionality reduction on the RW2 dataset. Figure 9.5 shows the average runtime for the three lower bounding filters on a logarithmic scale. Anticipatory pruning, denoted as AP, yields substantial runtime improvements compared with the base methods $LB_{Keogh}$, $FTW$, and $LB_{Hybrid}$. It also outperforms the early stopping approach (ES) [SYF05] for all three methods. Anticipatory pruning is especially helpful for strong reductions where it makes up for the loss in information of the filter step by pruning during DTW refinement. Figure 9.6 summarizes the gains by showing the relative improvement for the same experiment. To abstract from implementation issues, Figure 9.7(a) demonstrates that the runtime gains are due to a reduction in the number of required calculations.
9.4. Experiments

For the second synthetic dataset, a summary of relative improvement with respect to the required number of calculations is given in Figure 9.7(b). We observe similar gains in performance, where the difference for the $LB_{Hybrid}$ method is most pronounced and shows a reduction from around 75% (AP) to 40% (for early stopping).

On the NEWSVid data, the behavior is similar for $LB_{Keogh}$ and $LB_{Hybrid}$ (see Figure 9.7(c)). For FTW, however, we observe relatively little improvement for low reductions, but rapid improvement for strong reductions.

9.4.2 Univariate and Multivariate Time Series

Our next set of experiments evaluates the effect of the number of attributes on the performance of anticipatory pruning. Based on the results of the preceding section, a reduction to 16 dimensions was chosen. As depicted in Figure 9.8(a) for the RW2 dataset, the general tendency is the same as for the dimensionality reduction. The performance gain of anticipatory pruning even increases with the number of attributes. While anticipatory pruning avoids some DTW computations in the univariate case, this effect is much more pronounced for multivariate time series.

Figure 9.8(b) shows the same experiment for the RW1 dataset. For this dataset, univariate time series benefit considerably from anticipatory pruning and the behavior is fairly consistent over the evaluated range of attributes.
9.4.3 Bandwidth

In Figure 9.9(a), we study the influence of the bandwidth constraint on the NEWSVid dataset. Dimensionality reduction was again set to 16 dimensions. A remarkable reduction in the number of calculations can be observed for all three lower bounding filters. There is a slight decrease with respect to the bandwidth, but even for extremely wide bands of 150, anticipatory pruning yields substantial improvements. This effect is also present on the TREC Vid dataset (Figure 9.9(b)) and the SignLanguage dataset (Figure 9.9(c)).

9.4.4 Length of Time Series

Our next study empirically validates the scalability of anticipatory pruning with respect to the length of time series on three multivariate time series datasets. Because of the varying length, no fixed dimensionality reduc-
9.4. Experiments

Figure 9.10: Efficiency improvement (#calc.) for varying time series lengths on different databases.

As shown in Figure 9.10(b), the results are similar for the smaller SignLanguage dataset with sequences of length up to 512. Even though this dataset shows a jump in the pruning power of FTW for length 128, the performance gains of anticipatory pruning are robust.

On the third real world dataset, TRECVid, anticipatory pruning shows similar pruning power and scalability (see Figure 9.10(c)).
Figure 9.11: Efficiency improvement (#calc.) for varying numbers of nearest neighbors on NEWSVid.

### 9.4.5 Number of Nearest Neighbors

In our last experiment, we evaluate the effect of varying the number of nearest neighbor retrieved during query processing. This parameter has only very limited effect on the pruning capability of our technique as demonstrated in Figure 9.11 by the great performance gains of anticipatory pruning for the NEWSVid dataset.

### 9.5 Conclusion

In this chapter we investigated speeding up similarity search query processing under DTW. Our novel anticipatory pruning makes best use of a family of speed-up techniques based on multistep filter-and-refine architectures. By computing an estimated overall DTW distance from already available filter information, a series of lower bounds of the DTW is derived that requires hardly any overhead. This is in contrast to existing approaches, where the information that aDTW exploits is not used, resulting in many unnecessary DTW computations. Our technique can be flexibly combined with existing and future DTW lower bounds. In the experimental evaluation, we demonstrated a substantial reduction in the number of calculations and consequently a significantly reduced runtime. Since the computational overhead with respect to early stopping is negligible, aDTW should be the method of choice in most scenarios. This obtained efficiency gain can be used in situa-
tions where DTW is combined with more expensive ground distances, as for example the video copy detection approach presented at the end of Chapter 8.

In the next chapter, we use another type of information to accelerate DTW query processing. In the considered application scenario, several DTW queries need to be processed simultaneously. We develop an approach that exploits shared characteristics between these queries for processing them at the same time.
In many of today’s applications, large numbers of queries arise at any given time. Existing algorithms for efficient DTW query processing, however, are designed for single queries and cannot process multiple DTW queries simultaneously, a serious limitation which slows down overall processing.

In this chapter, we develop an efficient processing approach for multiple DTW queries. It is based on the observation that algorithms in areas such as data mining and interactive visualization incur many queries that share certain characteristics. Our solution exploits these shared characteristics by pruning database time series with respect to sets of queries, and we prove a lower-bounding property that guarantees no false dismissals. Our technique can be flexibly combined with existing DTW lower bounds or other single DTW query speed-up techniques for further runtime reduction. Our experiments demonstrate substantial performance gains for multiple DTW queries.

*This chapter has been published in the Proceedings of the 23rd International Conference on Scientific and Statistical Database Management (SSDBM 2011) [KGI’11].
10.1 Motivation

Today, vast numbers of queries need to be processed within limited time. Examples include interactive visualization [BS07], sensor networks [KLW+09], and data mining applications [BEKS00]. Fast response times are required for timely reaction to events, scalability, and interactivity. DTW is an effective measure for such domains, but it suffers from quadratic complexity in the length of the time series; therefore, fast query processing approaches have been proposed [KPC01, Keo02, ZS03, SYF05, SC04, AKAS08, AWK+09].

However, all of the existing speed-up techniques target single ad-hoc queries. We propose a novel method for combined processing of multiple DTW queries that achieves, compared to single-query approaches, substantial runtime improvements. It is based on a property of many multiple-query scenarios: queries may be similar or even share subsequences. As an example from data mining [BEKS00], consider density-based clustering where the transitive closure of the neighborhood (near-by objects) is computed to determine dense regions (clusters), resulting in similar queries for near-by objects. Another example is the detection of suspicious patterns presented in Chapter 8, where several patterns are queried simultaneously. More examples of related queries can be found in [BEKS00].

Our new approach exploits similarity among queries to process groups of queries combined as one, enabling the pruning of time series that are irrelevant to the entire group. To further enhance the pruning power of our concept, we introduce a nested hierarchy of query subgroups. This hierarchy iteratively splits the query group with respect to mutual similarity in order to continue pruning for smaller and more similar query subsets. We prove that this does not incur any false dismissals, i.e. we guarantee completeness of the query results. We show that our approach is orthogonal in nature to existing single DTW query approaches. We demonstrate how DTW filters for single query processing can be extended to multiple query processing and how they can be added to our query hierarchy. We introduce a tree structure that maintains all information necessary for efficient algorithmic processing of query (sub-)groups using any number of filters.
This chapter is structured as follows: In Section 10.2, we discuss related work on multiple query processing and DTW computation. Section 10.3 introduces our novel method for processing of multiple DTW queries. Section 10.4 presents experimental results and Section 10.5 concludes.

### 10.2 Comparison with Related Work

Multiple query processing has been studied in the context of query optimization [Sel88], where the goal is to devise efficient query execution plans. Similar problems have been studied for density-based clustering [YRW09], continuous queries on data streams [BW01, TB02], or in sensor networks [JMHA10]. While the concept of sharing computations among queries is widely used, these approaches are not designed for time series data.

Many approaches for efficient DTW query processing were introduced, and an overview was given in Section 9.2 of Chapter 9. All of these approaches are for single queries, i.e. they devise a filter-and-refine or approximate search mechanism that processes a single query at a time. For multiple queries, however, these techniques fail to exploit the potential that lies in sharing computations among time series queries.

The approach in [WKH05] matches single objects from a data stream to a set of queries, i.e. for each single object a result set of queries is obtained ($1 : n$ matching). Our approach, in contrast, differs in technique and application scenario: First, we apply query processing of several queries on a whole database; that is, we obtain a result set for each query with respect to the whole database ($n : m$ matching). Second, we can handle range queries with a different range for each query as well as $k$NN queries; this is only possible in scenarios where the whole database is used.

### 10.3 Efficient Multiple DTW Query Processing

We introduce our technique in several steps. In Section 10.3.1 we define the multiple DTW query and our general approach. Section 10.3.2 discusses the hierarchy and its representation in our multiple query tree. The combi-
nation with extended single query speed-up techniques is presented in Section 10.3.3. An algorithmic solution for processing of \(k\)NN queries is described in Section 10.3.4, and Section 10.3.5 discusses the special case of range queries with individual thresholds.

### 10.3.1 Multiple DTW Query

We begin by introducing the problem definition of processing multiple DTW queries. In this section, we discuss range queries. The extension to the more complex problem of \(k\)NN queries is presented in Section 10.3.4.

Processing multiple range queries corresponds to computing a result set for each query of all time series that are within DTW distance of \(\varepsilon\) from the respective query. Clearly, there might be applications where different thresholds apply per query, and this aspect is discussed in Section 10.3.5.

**Definition 10.1** Given a set of time series queries \(Q = \{q^1, \ldots, q^c\}\) and a time series database \(DB\), a **Multiple DTW \(\varepsilon\)**-query determines multiple result sets \(Res^i = \{t \in DB \mid DTW(q^i, t) \leq \varepsilon\}\) for all \(i = 1, \ldots, c\).

In terms of relational algebra, this corresponds to a join of the query set and the database: \(Q_{DTW \leq \varepsilon} DB\).

Each query has an individual result set, i.e. the different results are not merged in any way. Thus it is possible to process a multiple DTW query simply by dividing \(Q\) into \(c\) single, independent queries \(q^i\) such that traditional single query processing algorithms can be applied. This, however, would not exploit the speed-up potential that lies in the knowledge about the overall query set \(Q\). Our novel approach operates directly on the whole set \(Q\), i.e. the queries are examined simultaneously. This technique allows for sharing computations, since pruning of many time series can be done for the entire set at once, thereby speeding up the overall response time considerably.

We introduce the **multiple query distance function**, which uses a single calculation for a set of queries. Using this multiple query distance function, pruning of irrelevant time series is performed for all queries simultaneously, obtaining a substantial reduction of the calculations for the individual result
Figure 10.1: Multiple DTW query processing framework.

sets. As we will see later on, pruning is done such that false dismissals are avoided, thereby ensuring a complete result set. We then remove false alarms to guarantee correctness of the results.

The general processing scheme is illustrated in Figure 10.1: First, an intermediate result set based on our multiple DTW query distance $\text{multiDTW}$ is jointly determined for all queries. This shared set $\text{Res}^Q$ is the remainder after pruning the database $\text{DB}$ and is an approximation of the final, individual result sets $\text{Res}^i$. By design, we guarantee no false dismissals of the multiple query distance function: If $t \notin \text{Res}^Q$, then $t \notin \text{Res}^i$ for any result set. Accordingly, to reject a time series $t$, just a single calculation of the multiple query distance function is needed, instead of $|Q|$ individual calculations. Since the queries share some results, but also differ in some, the set $\text{Res}^Q$ needs further processing: $\text{Res}^Q$ is split up for each query individually and false alarms are removed, i.e. more time series are rejected to obtain the results $\text{Res}^i$.

To ensure that final results are indeed complete (i.e., no false dismissals), the intermediate set needs to be a superset of each exact result set. More formally, the condition $\text{Res}^Q \supseteq \text{Res}^i$ has to hold for all $i = 1, \ldots, c$. This is the case if the multiple query distance function fulfills the following property:

**Definition 10.2** A multiple query function $\text{multiDist}$ fulfills the **Shared Lower Bound Property** with respect to a distance function $\text{dist}$ iff

$$\forall Q, t: \forall q \in Q: \text{multiDist}(Q, t) \leq \text{dist}(q, t)$$

For each final result $t \in \text{Res}^i$ we have $\text{dist}(q^i, t) \leq \varepsilon$ and by Definition 10.2 $\text{multiDist}(Q, t) \leq \varepsilon$; thus, if $\text{Res}^Q = \{t \in \text{DB} \mid \text{multiDist}(Q, t) \leq \varepsilon\}$, completeness is guaranteed.
The tightest multiple query distance function possible for DTW still fulfilling this property is \( \text{multiDist}(Q, t) = \min_{q \in Q} \text{DTW}(q, t) \). This, however, entails the calculation of \(|Q|\) many DTW distance values; a performance gain is not realized. To ensure an efficient processing, we need a compact, single representation that can efficiently be processed; therefore we aggregate the set \( Q \) by a multiple query bounding box.

**Definition 10.3** Given a query set \( Q \), the **Multiple Query Bounding Box** is a sequence

\[
[ (L_1, U_1), \ldots, (L_n, U_n) ] = [ B_1, \ldots, B_n ] = \text{multiBox}(Q)
\]

with \( L_i = \min_{q \in Q} q_i \) and \( U_i = \max_{q \in Q} q_i \) (where \( q_i \) is the value of time series \( q \) at time step \( i \)).

By using the minimal and maximal values per time step, this bounding box approximates the whole query set. This approximation has the benefit of being very compact, while preserving much information to ensure effective pruning. Based on this bounding box definition, we introduce our novel multiple DTW query distance function.

**Definition 10.4** The **Multiple DTW Query Distance Function** between a multiple query bounding box \( [B_1, \ldots, B_n] \) and a time series \( t \) with respect to a bandwidth \( k \) is defined as:

\[
\text{multiDTW}(Q, t) = \text{multiDTW}([B_1, \ldots, B_n], [t_1, \ldots, t_m]) =
\]

\[
\text{dist}_{\text{band}}(B_n, t_m) + \min \left\{ \begin{array}{ll}
\text{multiDTW}([B_1, \ldots, B_{n-1}], [t_1, \ldots, t_{m-1}]) & \\
\text{multiDTW}([B_1, \ldots, B_n], [t_1, \ldots, t_{m-1}]) & \\
\text{multiDTW}([B_1, \ldots, B_{n-1}], [t_1, \ldots, t_m]) & \\
\end{array} \right.
\]

with \( \text{dist}_{\text{band}}(B_i, t_j) = \begin{cases} \text{dist}(B_i, t_j) & \text{if } |i - j| \leq k \\ \infty & \text{else} \end{cases} \)

and \( \text{dist}(B_i, t_j) = \text{dist}((L_i, U_i), t_j) = \begin{cases} |t_j - U_i| & \text{if } t_j > U_i \\ |t_j - L_i| & \text{if } t_j < L_i \\ 0 & \text{otherwise} \end{cases} \)

\( \text{multiDTW}(\emptyset, \emptyset) = 0, \quad \text{multiDTW}(x, \emptyset) = \text{multiDTW}(\emptyset, y) = \infty \)
10.3. Efficient Multiple DTW Query Processing

Since the distance to the bounding box is used during the DTW calculation, we simultaneously consider all queries and the computational complexity is independent of the actual number of represented queries. We now prove that the shared lower bound property holds, i.e. using $multiDTW$ distance entails no false dismissals.

**Theorem 10.1** $multiDTW$ fulfills the shared lower bounding property with respect to $DTW$.

**Proof 10.1** We need to prove that $\forall p \forall q \in Q : multiDTW(Q, p) \leq DTW(q, p)$

It suffices to show that $\forall i, j : dist(B_i, p_j) \leq dist(q_i, p_j)$ with $q \in Q$ and $B = multiBox(Q)$. Then, no other alignment can be found by $multiDTW$ that would break Theorem 10.1. The proof is done by cases: We distinguish (a) $L_i \leq p_j \wedge p_j \leq U_i$, (b) $p_j > U_i$, and (c) $p_j < L_i$.

For (a) it holds since $dist(B_i, p_j) = 0$. For (b) the following applies: $dist(B_i, p_j) = |p_j - U_i|$ and $dist(q_i, p_j) = |p_j - q_i|$. Accordingly, we have to show that $|p_j - U_i| \leq |p_j - q_i|$. Since $p_j > U_i$ and $U_i \geq q_i$ (Def. 10.3) this equals to $p_j - U_i \leq p_j - q_i$. By subtracting $p_j$ and multiplying with $-1$ we obtain $U_i \geq q_i$, which is true according to Def. 10.3. The proof of (c) is analogue to (b).

10.3.2 Hierarchical Multiple DTW Query

Our multiple DTW query achieves a speed-up compared to single query processing by pruning of whole query groups. In this section, we explore further pruning through the creation of several subgroups. A single group implies that only a single multiple query needs to be processed, which helps reducing the number of DTW computations but still might lead to a relatively large intermediate result set $Res^Q$. To prevent false alarms, we would need to compare each individual query against this intermediate set. We propose to reduce computations at this point by splitting up the query group into smaller subgroups for further pruning. This hierarchy of pruning options is based on the observation that smaller, more similar groups reduce the intermediate result size since the distances for smaller groups are larger, which is
Simultaneous Processing of Multiple DTW Queries

caused by tighter bounding boxes. The relation of group size and distance is
reflected by the following theorem.

**Theorem 10.2** Smaller query sets correspond to a more accurate multiDTW,
i.e. the distances values are larger:

\[ \forall Q, Q' \subseteq Q, p: \text{multiDTW}(Q, p) \leq \text{multiDTW}(Q', p) \]

**Proof 10.2** Similar to the proof of Theorem 10.1, we need to show that \( \forall i, j: \)
\[ \text{dist}(B_i, p_j) \leq \text{dist}(B'_i, p_j) \text{ with } B = \text{multiBox}(Q) \text{ and } B' = \text{multiBox}(Q'). \]
This ensures that no other alignment can be found by multiDTW that would violate the above theorem, i.e. that the left term in the inequation would be larger than the right term. The proof has three cases: (a) \( L_i \leq p_j \land p_j \leq U_i \), (b) \( p_j > U_i \), and (c) \( p_j < L_i \).

For (a) \( \text{dist}(B_i, p_j) = 0 \), \( \text{dist}(B_i, p_j) \leq \text{dist}(B'_i, p_j) \) is obviously fulfilled.
For (b) \( \text{dist}(B_i, p_j) = |p_j - U_i| \). Since \( U_i \geq U'_i \) (and thus \( p_j > U'_i \)) we also have
\[ \text{dist}(B'_i, p_j) = |p_j - U'_i| \]. We have to show that \( |p_j - U_i| \leq |p_j - U'_i| \). Since \( p_j > U_i \) and \( U_i \geq U'_i \) (Def. 10.3), we obtain \( p_j - U_i \leq p_j - U'_i \). By subtracting \( p_j \) and multiplying with \(-1\) we get \( U_i \geq U'_i \). According to Def. 10.3 this yields \( \max_{q \in Q} q_i \geq \max_{q \in Q'} q_i \).
This holds because of \( Q \supseteq Q' \).

Thus, reducing the query set corresponds to increasing the multiDTW distance
and thereby the number of time series that can be pruned.

In principle, multiDTW can be used on each partition of an arbitrary
partitioning of the query set \( Q \). In the most extreme case of \( |Q| \) many parti-
tions, it degenerates to single query processing. With fine grained groups
we get small intermediate result sets; however, at high computational costs
since many queries need to be processed. On the other hand, just one query
group is computationally efficient; however, we get a larger set of candidate
objects. We propose to combine different granularities through a hierarchi-
cal grouping of the query set. We organize queries in our multiple query tree,
where the root node represents all queries as a single group (cf. Figure 10.2),
and by descending into the tree we get more fine grained groups. Eventually,
the exact DTW computation for single queries is represented at leaf level.
Definition 10.5 Given a query set $Q$, the Multiple Query Tree (MQ-Tree) is defined as

- the root node stores a multiple query bounding box for all queries $Q$
- each inner node stores a multiple query bounding box for a query subset $P \subseteq Q$
- each leaf node stores the time series of a single query $q \in Q$
- all child nodes of a parent represent a complete, disjoint partitioning of their parent, i.e., $P = \bigcup_{P' \in \text{children}(P)} P'$
- the granularity is refined on each level, i.e., $\forall P' \in \text{children}(P) : P' \subset P$

During query processing, each time series $t \in DB$ is first checked against the root node, i.e. the whole query set. If the $\text{multiDTW}$ distance is already larger than the range threshold $\varepsilon$, the time series is pruned. Otherwise the child nodes are analyzed, i.e. subsets of all queries are considered. Again, $t$ is potentially pruned for some of the child nodes or passed on to the next level. Eventually, leaf nodes may be reached. If for the current leaf node/single query the DTW distance is still smaller than $\varepsilon$, $t$ is added to the corresponding result set. The completeness of this method is ensured because a) for each query $q^i$ there exists only one path from the root node to the corresponding leaf, b) all query sets along this path contain the query $q^i$, and c) the corresponding distance function along this path fulfills the shared lower bounding property.
To find query subgroups that are similar and therefore result in many pruned time series, we use clustering. The hierarchical query groupings of our tree are obtained with OPTICS [ABKS99], a hierarchical extension of the density-based clustering algorithm DBSCAN. Density-based clustering is most suitable for our approach because it ensures compact bounding boxes. Other clustering paradigms could, for example, assign outliers to query groups, yielding non-compact bounding boxes reducing the pruning potential. OPTICS requires no knowledge about the number of clusters a priori, enabling a flexible fan-out in our tree that adjusts automatically to the mutual similarity of queries. OPTICS computes a plot of possible clusters at different levels of density. Any horizontal cut in the plot corresponds to a clustering in this hierarchy. To generate our tree, we first obtain a list of all cuts at which a change in the number of clusters occurs (compared to the preceding cut). From this ordered list, which begins with two clusters, we choose enough cuts such that their combination corresponds to a cluster hierarchy of a given maximal depth. In case there are not enough cuts to reach the maximal depth, all cuts are used. Each level of the obtained cluster tree represents a (possible) non-horizontal cut through the OPTICS plot. In our experiments, we chose a maximal depth of 4. OPTICS has two parameters: The maximum density neighborhood range $\varepsilon'$, and the minimal number of objects in this neighborhood $\text{minPts}$. We set $\varepsilon'$ to twice the $\varepsilon$-range of our multiple DTW query and $\text{minPts}$ to 2 for avoiding single queries in inner nodes. Since the number of queries is typically small compared to the number of time series in the database, clustering runtimes are negligible, as our experiments in Section 10.4 confirm.

10.3.3 Filter-supported Hierarchical Multiple DTW Query

Our hierarchical multiple DTW technique makes use of similarity among multiple queries, and can exploit a hierarchy of pruning possibilities that greatly reduce DTW computations. Existing work for single queries accelerate DTW computations by using lower-bounding filter functions in filter-and-refine algorithms. Since several lower bounds exist in the literature, one can
10.3. Efficient Multiple DTW Query Processing

Figure 10.3: Sequence of traditional filter-and-refine steps for single queries: each query \((q^1, q^2, \ldots)\) undergoes the same filter steps until it is pruned or refined.

build a sequence of filters that a query can be subjected to, as illustrated in Figure 10.3.

Clearly, this filter concept is orthogonal to our multiple DTW query. Our approach prunes based on the combination of multiple queries into a single representation, whereas the traditional approach uses filters per query. For improved pruning power and thereby most efficient query processing, we propose a combination of the two concepts, which we call filter-supported hierarchical multiple DTW.

To realize such a combination, the single query lower bound needs to be extended so that it can serve as a lower bound for a query group, and we go from a traditional filter distance function \(\text{dist}_f(q, p)\) to a multiple query distance function \(\text{multiDist}_f(Q, p)\). In this work we exemplarily adapt the linearization-based \(\text{LB}_{\text{Keogh}}\) lower bounding filter to handle multiple queries [Keo02], which is based on the difference to upper and lower bounds (termed “envelope”) within the band constraint (cf. Sec. 9.3.7 on page 171).

Traditionally, \(\text{LB}_{\text{Keogh}}\) between a query \(q\) and time series \(t\) is defined as

\[
\text{LB}_{\text{Keogh}}(q, t) = \sum_{i=1}^{n} \begin{cases} 
|t_i - \hat{U}_i| & \text{if } t_i > \hat{U}_i \\
|t_i - \hat{L}_i| & \text{if } t_i < \hat{L}_i \\
0 & \text{otherwise}
\end{cases}
\]

with bounds \(\hat{U}_i = \max_{(q_{i-k}, \ldots, q_{i+k})} \) and \(\hat{L}_i = \min_{(q_{i-k}, \ldots, q_{i+k})} \) w.r.t. DTW bandwidth \(k\).

For a query group, we define the multiple query \(\text{LB}_{\text{Keogh}}\), which computes the upper and lower bounds for all queries within the band constraint using the previously introduced multiple query bounding box.
Definition 10.6 Given a query set \( Q \), the corresponding multiple query bounding box \( [(L_1, U_1), \ldots, (L_n, U_n)] \) and the DTW bandwidth \( k \), the Multiple Query LB\(_{Keogh}\) is defined as

\[
\text{multiLB}_{Keogh}(Q, t) = \sum_{i=1}^{n} \begin{cases} 
|t_i - \tilde{U}_i| & \text{if } t_i > \tilde{U}_i \\
|t_i - \tilde{L}_i| & \text{if } t_i < \tilde{L}_i \\
0 & \text{otherwise}
\end{cases}
\]

with upper bound over all queries \( \tilde{U}_i = \max_{q \in Q} \{U_{i-k} \ldots U_{i+k}\} \) and lower bound \( \tilde{L}_i = \min_{q \in Q} \{L_{i-k} \ldots L_{i+k}\} \).

To show that this multiple query LB\(_{Keogh}\) can be used without incurring false dismissals, we prove the shared lower bounding property.

Theorem 10.3 multiLB\(_{Keogh}\) fulfills the shared lower bounding property with respect to DTW.

Proof 10.3 We need to show that for any query set \( Q \) and any time series \( t \), we have \( \text{multiLB}_{Keogh}(Q, t) \leq \text{DTW}(q, t) \).

According to Def. 10.3, the upper bound \( \tilde{U}_i \) for query set \( Q \) is the maximum of the upper bounds for all \( q \in Q \). Analogously, the lower bound \( \tilde{L}_i \) is the minimum of the lower bounds for all \( q \in Q \). Thus, similar to Theorem 10.1’s proof, we know that for an individual query \( q \) in \( Q \) the bound can only be higher than for the entire set, i.e. we have that \( \text{multiLB}_{Keogh}(Q, t) \leq \text{LB}_{Keogh}(q, t) \forall q \in Q \). Since \( \text{LB}_{Keogh} \) is itself a lower bound to DTW [Keo02], i.e. \( \text{LB}_{Keogh}(q, t) \leq \text{DTW}(q, t) \), it holds that \( \text{multiLB}_{Keogh}(Q, t) \leq \text{DTW}(q, t) \forall q \in Q \).

Accordingly, our definition of the multiLB\(_{Keogh}\) filter may be safely combined with our hierarchical multiple DTW approach without jeopardizing completeness. Figure 10.4 gives an overview: as before, we have different granularities of the query subgroups \( Q', Q'', \ldots \) of the entire query set. Additionally, sequential filters, such as dist\(_{f1}\) in Figure 10.3, extended to multiple queries as multiDist\(_1\) in Figure 10.4 are added (e.g., LB\(_{Keogh}\) to multiLB\(_{Keogh}\)). In Figure 10.4, we illustrate a complete nesting of both paradigms, i.e. for each granularity all possible sequential filters are used.
In general, on each level it is possible to select a certain subset of filter distance functions. For example, we could omit the gray highlighted nodes of the tree and directly propagate the result to the next level. In this manner, it is possible to flexibly construct a subgroup and filter cascade based on selectivity estimates.

We extend the multiple query tree (Def. 10.5) to include sequential filters in addition to grouping of queries. is defined as follows:

**Definition 10.7** Given a query set \( Q \) and a series of multiple query distance functions \( \text{multiDist}_i \) with \( i = 1, \ldots, r \) (fulfilling the shared lower bound property w.r.t. DTW), the **Filter-supported Multiple Query Tree (FSMQ-Tree)** is a modified MQ-Tree:

- the root node is a tuple \([Q, 1]\) of all queries \( Q \) and the distance function \( \text{multiDist}_1 \)
- each inner node is a tuple \([P, i]\) representing a query set \( P \subseteq Q \) and the distance function \( \text{multiDist}_i \)
- each leaf node is a tuple \([\{q\}, r + 1]\) representing a single query \( \{q\} \subseteq Q \) and the usual distance function DTW
- all child nodes of a parent represent a complete, disjoint partitioning of their parent, i.e., \( P = \bigcup_{[P', j] \in \text{children}([P, i])} P' \)
- either the granularity or the distance function is refined, i.e., \( \forall [P', j] \in \text{children}([P, i]) : P' \subset P \) or \( j > i \)

Each node of the tree now represents a query subset and a filter function. The hierarchy is used to either go to a new granularity by splitting the query group, or to go to a new lower bounding filter. Since the leaves use DTW itself, and we assume that all filters are lower bounding, the final result is correct and complete.

The query processing is illustrated in Algorithm 10.1. For a query set \( Q \), a database \( DB \) and an \( \varepsilon \)-threshold, we begin by initializing an empty result set for each query. Query processing for database time series starts from the tree root, which represents the entire query set \( Q \) and the first filter distance.
While we are not at leaf level (line 6), we compare the current multiDist value to the $\varepsilon$-threshold (line 10). We continue processing only for child nodes that cannot be pruned according to this filter (line 12). At leaf level, the final DTW refinement determines whether a time series is included in the respective result set of query $q_i$ (line 8). Thus, the FSMQ-Tree maintains all information needed on the current query subset and the respective filter level, making it easy to prune time series that do not contribute to certain result sets.

### 10.3.4 Multiple DTW $k$NN-Query

In the previous sections we discussed processing of multiple range queries. In range queries, the pruning threshold is known to be exactly the $\varepsilon$-range. We now discuss the case of multiple $k$NN ($k$ nearest neighbor)-queries, which is more complex in the sense that such a threshold is not known in advance. The result set consists of the $k$ most similar time series.

**Definition 10.8** Given a set of time series queries $Q = \{q^1, \ldots, q^c\}$ and a time series database $DB$, a **Multiple DTW $k$NN-query** determines multiple result sets $Res^i(DB)$, such that $|Res^i(DB)| = k$ and $\forall t \in Res^i(DB) \forall s \in DB \setminus Res^i(DB) : DTW(q^i, t) \leq DTW(q^i, s)$ for all $i = 1, \ldots, c$. 
10.3. Efficient Multiple DTW Query Processing

**Algorithm 10.1**: Processing of multiple DTW $\varepsilon$-queries.

1. **input**: FSMQ-Tree with root $[Q, 1]$, $\varepsilon$ range, database $DB$
2. **foreach** $t \in DB$ **do**
   3. $\text{process-node}([Q, 1], t)$
4. **return** result sets $Res^i$, $i = 1, \ldots, |Q|$

5. **function** $\text{process-node}(node [P, j], \text{time series } t)$
   6. **if** $(P = \{q^i\} \land j = r + 1)$ **then**
      7. **if** $(\text{DTW}(q^i, t) \leq \varepsilon)$ **then**
         8. $Res^i = Res^i \cup \{t\}$
   9. **else**
      10. **if** $(\text{multiDist}_j(P, t) \leq \varepsilon)$ **then**
         11. **foreach** $[P', l] \in \text{children}([P, j])$ **do**
              12. $\text{process-node}([P', l], t)$
13. **end function**

In contrast to range queries, where the maximal permitted distance $\varepsilon$ is known beforehand, for $k$NN-queries such a threshold is not given a priori. This threshold, however, plays a crucial role for the early pruning of time series in our hierarchy. To obtain a threshold for $k$NN-queries, query processing typically retrieves $k$ objects that are iteratively replaced by closer ones as query processing proceeds. The distance of the $k$th best candidate seen so far acts as a moving $\varepsilon$-threshold. At the end of query processing, it corresponds to the true $k$th nearest neighbor distance.

For multiple $k$NN-queries, the question is how to obtain an initial $\varepsilon$-threshold that is as small as possible. If the threshold is too large, only few objects are pruned resulting in larger intermediate result sets and, thus, in higher query times.

Naively, one could process $k$ time series for the group of queries, thus obtaining an initial $\varepsilon$-threshold that corresponds to the maximum $k$th nearest neighbor distance for all queries. Then, as more time series are processed, the $k$ nearest neighbors of the query set could be adjusted to potentially lower the moving threshold for the query set accordingly. This procedure, however, is not the most efficient one. Since we use a hierarchical grouping
of queries, maintaining only a single moving threshold for the entire query set is only useful at root level. Descending into the tree, i.e. using more fine grained partitions, allows calculating tighter thresholds by considering only the queries represented by the current subtree.

As solution, we maintain a moving threshold per query subgroup for maximizing pruning power: each tree node is enriched by an individual $\varepsilon$-value (i.e., the $k$th nearest neighbor distance among the time series seen so far).

**Definition 10.9** Given a query set $Q$ and a series of multiple query distance functions $\text{multiDist}_i$ with $i = 1, \ldots, r$ (fulfilling the shared lower bound property w.r.t. DTW), the $\varepsilon$-enriched FSMQ-Tree for currently completely processed time series subset $T \subseteq \mathcal{D}$ is defined by:

- each node is a 3-tuple $[P, i, \varepsilon]$
- a valid FSMQ-Tree (cf. Def. 10.7) is obtained if the 3-tuples are restricted to the first two components
- each inner node stores $[P, i, \varepsilon]$ with $\varepsilon = \max_{q^j \in P} \max_{t \in \text{currRes}_j(T)} \{\text{DTW}(q^j, t)\}$ and $\text{currRes}_j(T)$ is $q^j$’s current $k$NN query result for subset $T \subseteq \mathcal{D}$ (cf. Definition 10.8).

The individual moving threshold $\varepsilon$ is the maximum among the $\varepsilon$ values of its child nodes, which we use for efficient updating during query processing.

**Theorem 10.4** For each inner node of the $\varepsilon$-enriched FSMQ-Tree $[P, i, \varepsilon]$ the following holds: $\varepsilon = \max_{[P', i', \varepsilon'] \in \text{children}([P, i, \varepsilon])} \{\varepsilon'\}$.

**Proof 10.4** By construction of the FSMQ-Tree as modified MQ-Tree (Def. 10.7), it holds that $P = \bigcup_{P' \in \text{children}(P)} P'$ for all child nodes $[P', i', \varepsilon']$ of node $[P, i, \varepsilon]$. Consequently, $\varepsilon = \max_{q^j \in P} \max_{t \in \text{currRes}_j(T)} \{\text{DTW}(q^j, t)\} = \max_{q^j \in \bigcup_{P' \in \text{children}(P)} P'} \max_{t \in \text{currRes}_j(T)} \{\text{DTW}(q^j, t)\} = \max_{[P', i', \varepsilon'] \in \text{children}([P, i, \varepsilon])} \{\varepsilon'\}$ since $\varepsilon' = \max_{q^j \in P'} \max_{t \in \text{currRes}_j(T)} \{\text{DTW}(q^j, t)\}$. (Note that the maximum over the current result set $\text{currRes}_j$ does not change for a given $q^j$ and $T$.)
Accordingly, as we process time series and include them in the respective result sets of individual queries at leaf level, we can propagate the thresholds up the tree by taking the maximum among child nodes.

Algorithm 10.2: Processing of multiple DTW $k$NN-queries.

1. **input:** $\varepsilon$-enriched FSMQ-Tree with root $[Q, 1, \varepsilon_1], k, DB$
2. **foreach** $t \in DB$ **do**
   3. **process-node**($[Q, 1, \varepsilon_1], t$)
4. **return** result sets $Res^i$, $i = 1, \ldots, |Q|$  

Function **process-node**($node [P, j, \varepsilon]$, time series $t$)

5. **if** ($P = \{q^i\} \land j = r + 1$) **then**
   6. **if** ($DTW(q^i, t) < currEps^i$) **then**
      7. $currRes^i = currRes^i \backslash \{\arg\max_{x \in currRes^i} \{DTW(q^i, x)\}\}$
      8. $currEps^i = \max_{t \in currRes^i} \{DTW(q^i, t)\}$
      9. propagate $currEps^i$
   10. **else**
      11. **if** ($multiDist_j(P, t) \leq \varepsilon$) **then**
         12. **foreach** $[P', l, \varepsilon'] \in \text{children}(\{P, j, \varepsilon\})$ **do**
            13. **process-node**($[P', l, \varepsilon'], t$)
   14. **end function**

For a query set $Q$, a database $DB$, and the number of nearest neighbors $k$, we begin by initializing with $k$ randomly selected time series from the database to fill the result sets; i.e. at this stage, each result set contains the same set of objects. Afterwards the current individual $\varepsilon$ thresholds are calculated based on these sets and the $\varepsilon$-enriched FSMQ-Tree is built. The overall processing algorithm of multiple $k$NN queries on this tree is shown in Algorithm 10.2. As in Algorithm 10.1, in inner nodes database time series are pruned if they exceed the $\varepsilon$ threshold (line 13). There are two major differences in the query processing for multiple DTW $k$NN queries. First, pruning takes place according to a subgroup-specific threshold, which is only valid for the query subgroup represented by this inner node. Second, if a database time series reaches leaf level (line 6) and the object cannot be pruned by
the current individual threshold \( \text{currEps}^i \), an update of the corresponding result list is performed. The time series with the highest DTW distance to the query is substituted by this more similar time series (lines 8-9). Such an update may lead to improved pruning for the remaining database time series, because the threshold on the \( k \)th neighbor distance among time series seen so far for \( q^i \), i.e. \( \text{currEps}^i \), becomes smaller (line 10). Note that the update of thresholds in line 10 is based on stored values and not on re-computation of the DTW distances. The better threshold is propagated up the tree by recomputing the maxima according to Theorem 10.4, improving pruning on all levels. Additionally, we improve the tree structure by adjusting the query grouping. As mentioned in the discussion of range queries (cf. Section 10.3.2), OPTICS clustering is used to group queries. If the thresholds change, we refer to the hierarchical plot that OPTICS has generated to re-group the queries. This ensures that we always group the queries based on their similarity with respect to the current pruning options.

In the following, we prove that this multiple \( k \)NN query processing can be used without incurring false dismissals, i.e., it ensures completeness of the results.

**Theorem 10.5** Query processing of multiple \( k \)NN-queries using the \( \epsilon \)-enriched FSMQ-Tree is complete, i.e. there are no false dismissals.

**Proof 10.5** Time series are only dismissed based on the pruning thresholds in the \( \epsilon \)-enriched FSMQ-Tree nodes. False dismissals are prevented if the pruning threshold for any query is larger than or at most equal to the final \( k \)th nearest neighbor distance [Fal96, SK98]. This means that completeness is ensured if in all (inner and leaf) nodes of the tree the subgroup-specific thresholds \( \epsilon \) are larger than or equal to the exact thresholds of any single query at leaf level in the corresponding subtree. This means that for each node \([P, i, \epsilon]\) it holds that

\[
\epsilon \geq \max_{q^i \in P} \{\max_{t \in \text{Res}^i(T)} \{\text{DTW}(q^i, t)\}\} \forall T \subseteq DB.
\]

Since in the inner nodes, the corresponding \( \epsilon \) can be determined by taking the maximum of the child nodes (cf. Theorem 10.4), it suffices to prove the above property for the leaf nodes; then it holds for the entire tree.
10.4. Experiments

In the leaf nodes, the query set consists of a single query \( q^i \), i.e. we have 
\[
\varepsilon = \max_{t \in \text{currRes}^i(T)} \{ DTW(q^i, t) \}.
\]
Since the currently processed time series are a subset of the database \( T \subseteq DB \), we have 
\[
\varepsilon \geq \varepsilon' := \max_{t \in \text{Res}^i(DB)} \{ DTW(q^i, t) \}
\]
which is the final \( k \)th nearest neighbor distance.

10.3.5 Multiple Range Queries with Individual Ranges

By augmenting our tree with the concept of individual \( \varepsilon \) values, we enable the processing of multiple \( k \)NN queries. Additionally, this allows processing of multiple DTW range-queries with individual thresholds \( \varepsilon^i \). In this case, the \( \varepsilon^i \) value of each leaf node is fixed, and the thresholds in inner nodes are determined at the beginning analog to Definition 10.9; i.e., for each inner node \([P, i, \varepsilon]\), the threshold \( \varepsilon \) is determined by 
\[
\max_{[P', i', \varepsilon'] \in \text{children}([P, i, \varepsilon])} \{ \varepsilon' \}.
\]
There is no dynamic change of the result set \( \text{currRes}^i \); it is just filled as database time series fall below the individual threshold \( \varepsilon^i \) (cf. Figure 10.1, line 10-11).

10.4 Experiments

Experiments were run on 3GHz Intel Core 2 CPUs using JAVA implementations. As mentioned in Section 10.3, all approaches use early stopping of DTW computations [KWX’06, SYF05]. Unless stated otherwise, the following defaults were used: Time series length was \( n = 512 \) and dataset size was 5,000. DTW bandwidth \( k \) was 10% of time series length. The number of nearest neighbors retrieved was 5 per multiple query (i.e., 5 per individual query), and for range queries an \( \varepsilon \)-range was selected that resulted in around 5 nearest neighbors per individual query. As measures we use the wall clock time averaged over 10 multiple queries, the relative number of refinements, i.e. the percentage of database time series that undergo exact DTW calculations, and the relative improvements of our method compared to the baseline method. A multiple query \( Q \) is obtained by randomly selecting a set \( S \) of seed queries from the database. For each seed \( s \in S \), we generate \( g \) queries deviating from the seed by a standard deviation of 10%, and a multiple query \( Q \)
has the cardinality $|Q| = |S| \cdot g$. As default, we use 8 seeds and 5 generated queries per seed, resulting in 40 individual queries per multiple query.

We use synthetic random walk (RW) and real world data. For RW, the $t_{i+1}$th value is generated by drawing from a normal distribution with parameters $\mu = 0$, $\sigma = 1$ added to the value of $t_i$: $t_{i+1} = t_i + N(0, 1)$. RW was normalized to an average value of 0. In real data experiments, we used datasets introduced in [Keo02]: From the EEG dataset we used 5,000 time series of length 512; the smaller datasets consist of 500 (burstin), 272 (lightcurb), and 180 (network) time series of length 100. The largest dataset EEG is used throughout the evaluation along with RW data, and an overview over the remaining results is given at the end.

There are no existing solutions for processing multiple DTW queries. As baseline method for comparison, we sequentially process the queries by employing a single-query filter-and-refine method, where the queries are processed independently using $LB_{Keogh}$ [Keo02]. We chose $LB_{Keogh}$ as the basis of our MultiDTW, but we could also have used other lower bounds as [SYF05, ZS03], since our approach is orthogonal to the concept of lower bounds. We use a simple sequential scan in our framework to process database time series, and our multiple-query technique can also be used with indexing techniques as [AKAS08] and dimensionality reduction to further speed up the overall runtime. It has been shown that for very high dimensional data such as time series, linear database scans outperform traditional index structures [WSB98].

The default configuration of our method is FSMQ as introduced in Section 10.3.3. In the FSMQ-Tree, we use $multiLB_{Keogh}$ (cf. Def. 10.6), the adaption of $LB_{Keogh}$ [Keo02], followed by $multiDTW$ (cf. Def. 10.4).

**Query Processing Strategy.** We begin our study by comparing the default configuration (FSMQ-Tree) to the earlier variants of our method on RW data in Figure 10.5. Shown are similar results for both range and $k$NN queries. The refinement percentages in the right figure validate that both MQ-Tree and the FSMQ-Tree dramatically reduce the number of DTW computations as opposed to the multiple DTW query. From the runtimes in the left figure, we can infer that even though the differences in the number of refinements
10.4. Experiments

Figure 10.5: Comparison of the three variants of our method on random walk data: The FSMQ-Tree (Section 10.3.3), the MQ-Tree (Section 10.3.2), and the simple multiple DTW query (Section 10.3.1).

Figure 10.6: Varying number of individual queries per multiple query.

are small, it is much more efficient to include filter support, as pruning can be performed much faster. Thus, the FSMQ-tTree is an efficient combination of the hierarchical refinement of multiple DTW queries with filter techniques.

Query Set Size and Similarity of the Queries. We investigate how the number of individual queries per multiple query affects the performance of the FSMQ-Tree and the single query processing method. Figure 10.6 shows the corresponding experiments; average query times (left y-axis) and relative improvements (right y-axis) are measured. The number of seeds per multiple query remains at the default value of 8. In (a) and (b) range queries and $k$-NN queries on random walk data are processed. For both query types, the average query times of single query $LB_{Keogh}$ increase much faster than the query times of FSMQ. This is confirmed by the relative improvements: FSMQ outperforms the single query solution significantly: for both range queries and $k$-NN queries we have relative improvements between 10 and 75 percent. It can be concluded that our method performs as intended, i.e. in situations of multiple queries a combined solution clearly surpasses an
independent solution. Similar conclusions can be made for the real world data in Figure 10.6(c), where the relative improvements go up to 87%. The corresponding relative numbers of refinements are shown in Figure 10.7(a). As one can see, the percentages are relatively stable for both approaches, but there is a substantial reduction in DTW computations achieved by the multiple query approach FSMQ.

In Figure 10.7(b) and 10.7(c) we show the effect of query similarity. We vary the number of similar queries per group, i.e. how many similar queries are generated for each seed of a multiple query. The absolute number of queries per multiple query is 40. In both experiments, the query times of our method improve, while the query times of the single query solutions are stable. The increasing relative improvements highlight this aspect.

**Database Size.** Figure 10.8 shows the performance for database sizes between 1,000 and 25,000 times series for range and kNN queries. FSMQ
outperforms single query processing for range queries in (a); the relative improvement is stable at around 55%, independently of the database size. With increasing database size, more time series fall within the $\varepsilon$-ranges of the queries. Accordingly, more exact DTW computations are necessary, compensating the positive effects of a larger database size. For the $k$NN queries, our approach copes with larger database sizes far better than single query processing, i.e. the relative improvement scales with the database size enabling performance gains of up to 70%.

**Time Series Length.** The influence of the time series length is shown in Figure 10.9. Since we assume the same length for query and database time series in this chapter, both are changed equally in these experiments. We used the random walk and EEG data. For both query types stable improvements of about 70% are achieved, independently of the time series length. Figure 10.9(c) demonstrates robustness of our approach for scattered values in the time series: the number of refinements required after applying the filter function $LB_{Keogh}$ goes up to nearly 100% of the database, which explains the large query times observed in Figure 10.9(b). This can be explained by the large variance in the EEG data: $LB_{Keogh}$ exploits the property that in most time series successive values are very similar by constructing a bounding box (envelope) for all values within the bandwidth constraint. Large scattering of the values leads to large boxes and thereby poor pruning power. Our approach, by contrast is not affected by this scatter, since queries are grouped by their similarity, and the hierarchy opens up pruning possibilities for subgroups as well.
Simultaneous Processing of Multiple DTW Queries

Figure 10.10: (a,b): Varying DTW bandwidth $k$. (c): Varying $\varepsilon$-range and varying number of NN.

Figure 10.11: Performance comparison on real world datasets burstin, lightcurb, and network.

**Bandwidth.** In Figure 10.10(a,b) we study the influence of the DTW bandwidth constraint $k$ on real world data. For both query types, very stable relative improvements of about 70-75% are achieved. While absolute runtimes increase for any approach, since increasing bandwidth means that more points in time are part of the DTW computation, the figures show that our approach reduces runtimes regardless of the bandwidth constraint.

**Number of Nearest Neighbors and the $\varepsilon$-range.** In Figure 10.10(c), we analyze how the parameters epsilon range and number of nearest neighbors of the two query types influence the performance. As for the bandwidth, it is clear that increasing these parameters will lead to higher computational cost, as larger result sets are to be expected. For both query types, FSMQ outperforms the single query processing method. This means that similar to the bandwidth experiment, while absolute runtimes increase, our FSMQ reliably reduces the runtimes by a considerable margin.
10.5. Conclusion

Additional Experiments on Real World Data. In Figure 10.11 we give an overview over the performance on other datasets also used in [Keo02]. For range queries, our FSMQ approach greatly reduces the number of DTW refinements necessary, and for $k$-NN queries, the performance gain of FSMQ over single query processing is even more pronounced.

10.5 Conclusion

In this chapter, we addressed the problem of multiple DTW queries. Our novel technique groups similar DTW queries for joint pruning of irrelevant time series in the database. By this, common characteristics between query time series are exploited. A hierarchy of subgroups of multiple DTW queries enables further pruning with iteratively smaller groups. We show that filter functions for single DTW queries can be extended to fit our multiple query processing approach. All information necessary to manage query groups, filter functions, and pruning thresholds is compactly represented in our Filter Supported Multiple Query Tree (FSMQ-Tree). We provide algorithms for efficient processing of range and $k$-NN queries on the FSMQ-Tree. Our experimental evaluation on synthetic and real world data demonstrates substantial runtime improvements compared to single DTW query processing.

With this chapter we conclude the fourth part of the thesis. We introduced DTW and presented two application scenarios, in which rapid processing of DTW is crucial. We contributed two novel techniques for accelerating DTW queries by using certain types of information: The aDTW approach exploits previously unused information from filter step for rapid rejection of candidates in the refinement step; the multiple DTW query approach uses shared characteristics of queries for simultaneous query processing. Both techniques are complementary, because aDTW could be integrated into the multiple DTW approach; at the root and the inner nodes piecewise filter information would be obtained for faster pruning in the lower levels of the query tree.
Part V

Summary
Chapter 11

Conclusion and Future Work

In this thesis, we contributed novel data mining and content-based similarity search techniques for temporal data. In data mining, we investigated effective methods for clustering of multivariate time series, for evaluation of stream clustering results, and for tracing of evolving subspace clusters in spatio-temporal data. In similarity search, we concentrated on new techniques for accelerating query processing under well-established distance measures. First, we proposed a technique for speeding-up processing of the Euclidean distance on non-temporal data, and then we developed new approaches for highly efficient time series similarity search under the Dynamic Time Warping distance. In this section, we review the main research results and give a brief overview over possible future research.

11.1 Conclusion

In the first part of this thesis, we developed a clustering method for multivariate time series, called Robust Time Series Clustering (RTSC). A major contribution of RTSC is the flexible cluster model, in which clusters are defined by sets of objects and individual sets of relevant intervals per dimension. In this subspace-clustering-based model, time series can be adaptively shifted on the time axis, and it allows for a certain fraction of deviating values in each relevant point in time. By this highly flexible model, complex patterns occurring in real world data can be effectively detected. Accord-
ingly, RTSC generates time series clusters of high quality, as it was confirmed by our experimental evaluation on synthetic and real world datasets.

In the second part, we proposed and investigated novel cluster mapping techniques. For the evaluation of stream clustering results, we contributed the Cluster Mapping Measure. In its core, it uses a mapping component based on class distribution histograms of clusters. These histograms allow a robust mapping in case of dynamic changes in the data distribution, as for example emerging and disappearing clusters. This mapping combined with a novel penalty concept enables effective evaluation of stream clustering results under different error settings, as we showed in an extensive experimental evaluation; our measure outperformed the nine competing evaluation measures. Next, we presented an approach for following evolving clusters in spatio-temporal data, for which we introduced the new paradigm of object-value-based cluster tracing. In this paradigm, clusters are traced according to their characteristics, independently of individual object identities. This is in contrast to existing approaches, which solely rely on common objects between clusters. The mapping component is based on a novel distance measure comprising both value and subspace similarity; the latter makes the approach highly suitable for complex data where patterns are hidden in subspaces. By applying the new concepts, our tracing approach allows the tracking of behavior over time, as for example general phenomena in oceanographic grid data.

In the third part of the thesis, we developed a novel indexing structure for similarity search under the Euclidean distance, which uses data mining methods in the index construction phase. Subspace clustering is used to identify local reductions, resulting in a high information content for each of the indexed objects. By a hierarchical nesting of local reductions we generate a multi-representation of objects such that queries have to traverse a cascade of different filters. Applying the train-and-test paradigm ensures compact region descriptions and adaptation to new data distributions. This novel combination of data mining methods and filter-and-refine in an indexing structure enables highly effective pruning, as we showed in our experiments on synthetic and real world data.
In the fourth part, we developed novel methods for fast processing of similarity search queries under the computationally expensive Dynamic Time Warping (DTW) distance. We presented two application scenarios, in which efficient processing of DTW queries is highly important. We introduced the novel concept of anticipatory pruning that makes best use of existing filter-and-refine architectures. Anticipatory pruning computes an estimated DTW distance from already available filter information, corresponding to a series of DTW lower bounds. These lower bounds eventually correspond to the exact DTW distance, and they cause nearly no computational overhead. Existing approaches, however, do not exploit the already existing information and incur many unnecessary DTW computations in the refinement step. In the experimental evaluation, we demonstrated that our methods achieves a substantial reduction in the number of calculations and, consequently, a much faster query processing. Next, we presented a technique that addresses multiple DTW queries, which exploits another type of information. Our method uses shared characteristics of query time series to obtain a nested hierarchy of query subgroups, which is used for processing subgroups of queries as a single query. This concept allows for pruning time series that are irrelevant to the entire group. We provide algorithms for efficient processing of range and \( k \)-NN queries, and our experimental evaluation showed significant runtime improvements compared to single DTW query processing.

**11.2 Future Work**

Based on the techniques and results contributed in this thesis, further challenging research objectives arise.

Our new indexing structure in Chapter 7 shows high performance on complex, non-temporal data. It is, however, not suitable for time series data since the underlying clustering algorithm is not designed for temporal data. For example, there is no concept that compensates misalignments on the time axis or that exploits the correlations between different points in time to
reduce the data dimensionality. Accordingly, combining the insights gained in the development of the indexing structure from Part III with our research results from Part I on time series subspace clustering is a promising research question. One possibility, for example, is to base an indexing structure on dimensionality reduced relevant intervals.

Future research on cluster mapping can go into several directions. In Chapter 5, we introduced the paradigm of object-value-based cluster tracing. Whereas being the first implementation of this paradigm, our approach has limitations corresponding to interesting research challenges. First, for clusterings of two successive points in time, cluster mapping is performed pairwise, ignoring the connections between clusters within the clusterings; therefore, impacts of other clusters that may affect the mapping are not considered. Accordingly, applying a global mapping approach like the Earth Mover's Distance, which would consider all clusters of two points in time simultaneously, is promising*. Second, mappings between two successive points in time are independent of the other time steps. Possible future work is to generate the mapping graph of all time steps based on global optimization considering all possible mappings and all possible points in time at once. Another research direction in the area of cluster tracing is to investigate other statistical representations of clusters that inherently cope with the temporal characteristics of data, for example based on dynamic Gaussian processes. Possible challenges adapting such approaches include the handling of complex cluster evolutions such as emerging or disappearing clusters.

*A vision paper discussing the challenges arising in global-mapping-based cluster tracing scenarios has been published in the Proceedings of the 25th International Conference on Scientific and Statistical Database Management (SSDBM 2013) [KGWS13].
Part VI
Appendices
Bibliography


[Agg05] Charu C. Aggarwal. On Change Diagnosis in Evolving Data Streams. IEEE Transactions on Knowledge and Data Engineering (TKDE), 17(5):587–600, 2005.


[AHWY03] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A Framework for Clustering Evolving Data Streams. In Proceedings of the International Conference on Very Large Data Bases (VLDB), pages 81–92, 2003.

[AHWY04] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A Framework for Projected Clustering of High Dimensional Data Streams. In Proceedings of the International Conference on Very Large Data Bases (VLDB), pages 852–863, 2004.


[BCD+07] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel,


BIBLIOGRAPHY


[HK01] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.


[LHY04] Yifan Li, Jiawei Han, and Jiong Yang. Clustering Moving Objects. In Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), pages 617–622, 2004.


[SYUK00] Yasushi Sakurai, Masatoshi Yoshikawa, Shunsuke Uemura, and Haruhiko Kojima. The A-Tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation. In *Proceedings of
the International Conference on Very Large Data Bases (VLDB), pages 516–526, 2000.


ence on Knowledge Discovery and Data Mining (KDD), pages 947–956, 2009.


Statement of Originality

This thesis would not have been possible without the close collaboration in the group of Professor Seidl. Many of the presented ideas and techniques evolved from numerous fruitful discussions in the group. While the high level of collaboration both within the group and with the students constitutes a key factor of the productive environment I found at Professor Seidl’s group, it makes it hard to pinpoint individual contributions. Professor Seidl is clearly to be credited with making numerous suggestions for improvements and additions to the presented work. The following provides some more detail on collaborations and support for the individual chapters.

The approach for subspace clustering of time series in Chapter 3, the cluster tracing techniques in Chapter 5, and the subspace-clustering-based index in Chapter 7 were developed by me and Stephan Günnemann; the approaches were analyzed in the diploma thesis’ of Arne Held, Charlotte Laufkötter, and Dominik Lenhard, respectively. The Cluster Mapping Measure in Chapter 4 was investigated in the diploma thesis of Timm Jansen, whom I advised together with Philipp Kranen. Special thanks go to Marc Wichterich for his help on the revision of the corresponding publication, which led to its ultimate acceptance. The work I did while being advised by Ira Assent and Ralph Krieger served as foundation for the Anticipatory DTW approach in Chapter 9, which was developed, implemented, and evaluated by me. Ira and Marc helped much in the formalization and presentation of the final ideas. The Multiple DTW query processing technique in Chapter 10 was developed in close collaboration with Stephan Günneemann and Anca Maria Zimmer (née Ivanescu).
List of Publications


