Advanced Analysis on Temporal Data

Johannes Aßfalg

München 2008

Advanced Analysis on Temporal Data

Johannes Aßfalg

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Johannes Aßfalg
aus Augsburg

München, den 19.05.2008

Erstgutachter: Prof. Dr. Hans-Peter Kriegel

Zweitgutachter: Prof. Dr. Daniel Keim

Tag der mündlichen Prüfung: 14.07.2008

Acknowledgement

This work would not have been possible without the support of so many people. I would like to take the opportunity to thank them for their contributions to this thesis.

First of all I would like to thank my supervisor Prof. Dr. Hans-Peter Kriegel. He gave me the opportunity to work with his group as an external PhD student. Not only has he shared his experience and knowledge with all members of his research group. He has also succeeded in creating a unique working atmosphere within the database group. I am also very grateful to Prof. Dr. Daniel Keim from the University of Konstanz who hasn't hesitated a second when asked to be the second referee for this thesis.

I owe much to my colleagues from the database group, especially the members of the time series team, in particular Dr. Peer Kröger, Dr. Peter Kunath, Dr. Matthias Renz, and Dr. Alexey Pryakhin. I enjoyed the interesting discussions with all of you. The database group was completed by Dr. Elke Achtert, Thomas Bernecker, Dr. Karsten Borgwardt, Dr. Stefan Brecheisen, Dr. Matthias Schubert, Stefanie Wanka, and Arthur Zimek. Thanks to all of you for a wonderful and interesting time.

I would like to thank Susanne Grienberger, who shouldered all the administrative work, so that we could focus on our research, thank you very much. Furthermore, I would like to extend my thanks to Franz Krojer. Franz was extremely helpful whenever problems occurred with the technical equipment or new hardware was required.

Aside from my colleagues from the university I would also like to thank my colleague from the sd&m AG in Munich. In particular I want to thank Dr. Marco Pötke who inspired my interest in similarity search on complex objects when I worked in his Geolus team. Thanks to all the my other sd&m colleagues for their patience when once again I was working for my thesis instead of being at the sd&m office.

Last but not least I would like to express my deepest thanks to my friends and to my family. My parents supported and encouraged me not only while working for this thesis, but throughout my whole life. Finally, I would especially like to thank Verena for her patience during the last months and for being there whenever I needed her.

Johannes Aßfalg,

Munich, May 2008

Abstract

Due to the increase in CPU power and the ever increasing data storage capabilities, more and more data of all kind is recorded, including temporal data. Time series, the most prevalent type of temporal data are derived in a broad number of application domains. Prominent examples include stock price data in economy, gene expression data in biology, the course of environmental parameters in meteorology, or data of moving objects recorded by traffic sensors.

This large amount of raw data can only be analyzed by automated data mining algorithms in order to generate new knowledge. One of the most basic data mining operations is the similarity query, which computes a similarity or distance value for two objects. Two aspects of such an similarity function are of special interest. First, the semantics of a similarity function and second, the computational cost for the calculation of a similarity value. The semantics is the actual similarity notion and is highly dependant on the analysis task at hand.

This thesis addresses both aspects. We introduce a number of new similarity measures for time series data and show how they can efficiently be calculated by means of index structures and query algorithms.

The first of the new similarity measures is threshold-based. Two time series are considered as similar, if they exceed a user-given threshold during similar time intervals. Aside from formally defining this similarity measure, we show how to represent time series in such a way that threshold-based queries can be efficiently calculated. Our representation allows for the specification of the threshold value at query time. This is for example useful for data mining task that try to determine crucial thresholds.

The next similarity measure considers a relevant amplitude range. This range is scanned with a certain resolution and for each considered amplitude value features are extracted. We consider the change in the feature values over the amplitude values and thus, generate so-called feature sequences. Different features can finally be combined to answer amplitude-level-based similarity queries. In contrast to traditional approaches which aggregate global feature values along the time dimension, we capture local characteristics and monitor their change for different amplitude values. Furthermore, our method enables the user to specify a relevant range of amplitude values to be considered and so the similarity notion can be adapted to the current requirements.

Next, we introduce so-called interval-focused similarity queries. A user can specify one or several time intervals that should be considered for the calculation of the similarity value. Our main focus for this similarity measure was the efficient support of the corresponding query. In particular we try to avoid loading the complete time series objects into main memory, if only a relatively small portion of a time series is of interest. We propose a time series representation which can be used to calculate upper and lower distance bounds, so that only a few time series objects have to be completely loaded and refined. Again, the relevant time intervals do not have to be known in advance.

Finally, we define a similarity measure for so-called uncertain time series, where several amplitude values are given for each point in time. This can be due to multiple recordings or to errors in measurements, so that no exact value can be specified. We show how to efficiently support queries on uncertain time series.

The last part of this thesis shows how data mining methods can be used to discover crucial threshold parameters for the threshold-based similarity measure. Furthermore we present a data mining tool for time series.

Zusammenfassung

Mit dem Voranschreiten der Entwicklung von Rechenleistung und ständig wachsenden Datenspeichern werden immer mehr Daten aller Art gespeichert, darunter auch temporale Daten. Diese Daten, insbesondere Zeitreihen, fallen in einer Vielzahl von Anwendungsbereichen an. Dazu zählen beispielsweise Börsendaten in der Wirtschaft, Gen-Expressionsdaten in der Biologie, Temperatur- und Luftverschmutzungsdaten in der Meteorologie oder Bewegungsdaten bei der Erfassung von Verkehrsströmen. Darüber hinaus lassen sich auch Daten aus dem Multimediabereich als temporale Daten auffassen, z.B. die aufeinanderfolgenden Töne eines Musikstücks.

Die immer größer werdende Menge an Rohdaten macht eine computergestützte Analyse dieser Daten mit Methoden des Data Minings unerlässlich, um anschließend potentiell nützliche Schlussfolgerungen basierend auf den Daten ziehen zu können. Eine der elementarsten Operationen aller Data Mining Verfahren ist die Ähnlichkeitsanfrage, die zwei Objekten einen Ähnlichkeitswert zuweist. Dabei sind zwei Aspekte von besonderer Bedeutung: die Semantik der zu definierenden Ähnlichkeitsfunktion und die Effizienz, mit der eine Ähnlichkeitsfunktion auf einer großen Menge an Daten berechnet werden kann. Die Semantik einer Ähnlichkeitsfunktion beschreibt, wann ein Objekt als ähnlich zu einem anderen Objekt betrachtet wird. Diese Semantik ist hochgradig abhängig von der Art der Anwendung.

Die Arbeit beschäftigt sich daher mit genau diesen beiden Aspekten. Zum einen werden verschiedene Verfahren zur Ähnlichkeitsbestimmung temporaler Daten eingeführt, zum anderen wird für die vorgestellten Methoden jeweils gezeigt, wie mit Hilfe von Indexstrukturen und geeigneten Anfragealgorith-

men das gewünschte Ergebnis effizient berechnet werden kann.

Das erste neue Ähnlichkeitsmaß basiert auf einem Grenzwert, den ein Benutzer dynamisch zur Anfragezeit vorgeben kann. Zwei Zeitreihen werden dann als ähnlich betrachtet, wenn der gewählte Grenzwert zu ähnlichen Zeiten über- und unterschritten wird. Nach der formalen Definition des Ähnlichkeitsmaßes zeigen wir anschließend, wie man Zeitreihen in eine Repräsentation überführt, die es erlaubt, Grenzwert-basierte Ähnlichkeitsanfragen effizient zu berechnen. Entscheidend dabei ist, dass der später verwendete Grenzwert nicht bekannt sein muss. Dies erlaubt beispielsweise auch die Entdeckung von besonders relevanten Grenzwerten mit Methoden des Data Mining.

Das nächste neue Ähnlichkeitsmaß basiert auf einem relevanten Amplitudenbereich. Der relevante Bereich wird mit einer bestimmten Auflösung abgetastet. Für jeden abgetasteten Amplitudenwert berechnen wir bestimmte Merkmale, sogenannte Features. Für jeden Featuretyp ergibt sich damit eine sogenannte Feature-Sequenz, die den Verlauf der Amplitudenwerte beschreibt. Da für jeden betrachteten Amplitudenwert mehrere Features bestimmt werden, ergeben sich schließlich mehrere Feature-Vektoren, die für die Bestimmung der Ähnlichkeit zweier Zeitreihen miteinander kombiniert werden. Entscheidend ist bei diesem Ähnlichkeitsmaß die amplitudenbasierte Abtastung der Zeitreihe im Gegensatz zu den herkömmlichen Feature-Werten, die entlang der Zeitachse abgeleitet werden und die Zeitreihe so auf globale Weise charakterisieren. Unsere Methode beschreibt den Verlauf von lokalen Feature-Werten. Schließlich kann bei unserer Methode ein relevanter Bereich von Amplitudenwerten angegeben werden und damit das Ähnlichkeitsmaß besser an die aktuellen Anforderungen angepasst werden.

Anschließend stellen wir die sogenannten Intervall-fokusierten Ähnlichkeitsanfragen vor. Dabei definiert der Benutzer einen oder mehrere Zeiträume, auf die sich der Anfragealgorithmus bei der Berechnung des Ähnlichkeitswertes beschränken soll. Das Hauptaugenmerk liegt bei diesem Anfragetyp auf der Effizienz. Es soll vermieden werden, dass die gesamte Zeitreiheninformation in den Hauptspeicher eingelesen werden muss, obwohl nur ein

relativ geringer Anteil der Zeitreihe benötigt wird. Wir schlagen daher eine Zeitreihenrepräsentation vor, die eine obere und eine untere Schranke für die tatsächliche Distanz liefert, so dass anschließend nur ein geringer Teil an kompletten Zeitreihen betrachtet werden muss. Auch hier muss der relevante Zeitabschnitt nicht schon im Vorfeld beim Befüllen der Datenbank bekannt sein, sondern kann zur Anfragezeit vom Benutzer vorgegeben werden.

Schließlich definieren wir ein Ähnlichkeitsmaß für sogenannte unsichere Zeitreihen. Dies sind Zeitreihen, für die zu jedem Zeitpunkt mehrere Werte vorliegen. Das ist dann der Fall, wenn entweder kein exakter Wert bestimmt werden konnte, oder absichtlich mehrere Messwerte auf einmal erfasst wurden. Auch für dieses neu definierte Ähnlichkeitsmaß beschreiben wir effiziente Speicherungs- und Zugriffsmethoden.

Im letzten Teil der Arbeit zeigen wir auf, wie mit Data Mining Methoden relevante Grenzwerte für einen gegebenen Datensatz bestimmt werden können und stellen ein Data Mining Tool für Zeitreihen vor.

Contents

Part I

Preliminaries

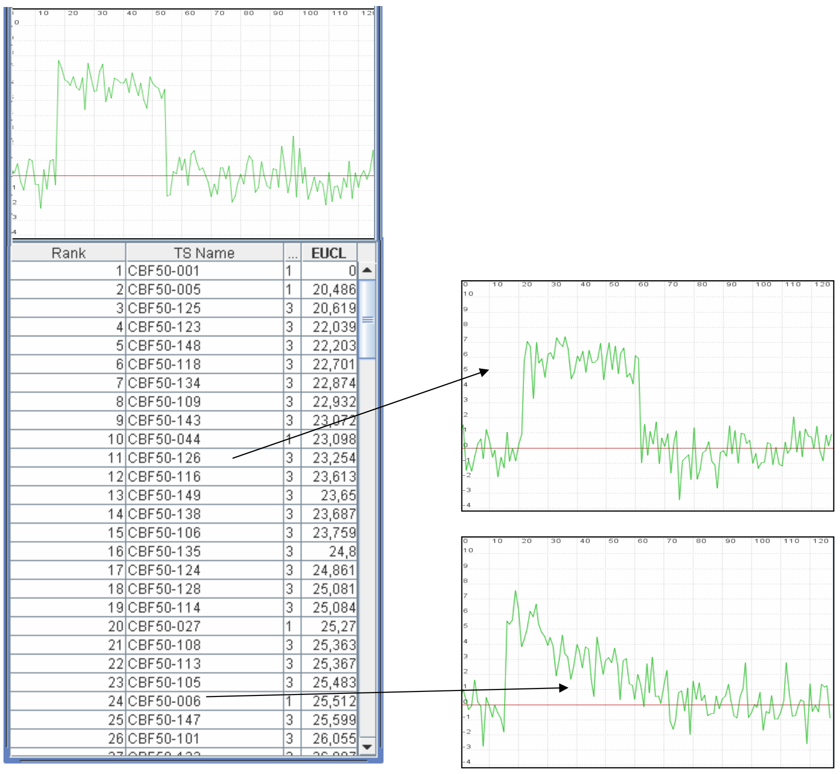
Chapter 1

Introduction

In probably all application fields, the amount of recorded and stored data increases tremendously. While simple data like strings or numbers can often be analyzed and retrieved with rather straightforward solutions, complex data usually requires a higher effort. Complex data is a large class of real-world data, including graphs, images, three-dimensional objects, or temporal data. In the following we give an overview of the importance of similarity measures for complex data.

1.1 Retrieval of Complex Objects

One of the basic tasks in databases is to retrieve objects that have been stored earlier. Usually, a query object is given, and the task is to find the objects of a dataset that are most similar to the query. This is the general principle of search engines. An example is depicted in Figure 1.1. A time series is used as the query object and the search engines returns an ordered list of similar time series. Depending on the used similarity measure, the ranking may vary significantly. So it is important to design similarity measures that are appropriate for a given task. Similarity measures try to model a certain notion of similarity, a similarity concept. Such a notion could be the human intuitive concept of visual similarity, or it could be the similarity with respect



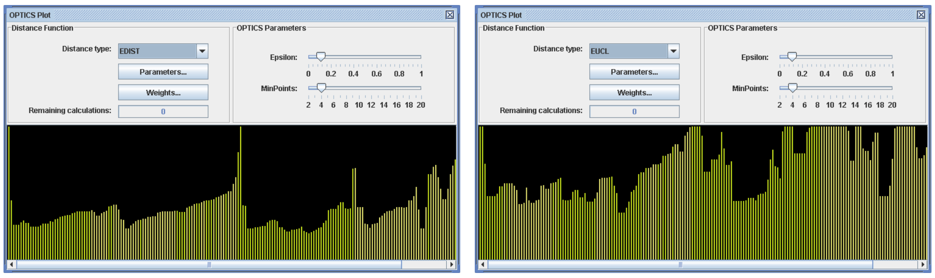
Rank	TS Name	...	EUCL
1	CBF50-001	1	0
2	CBF50-005	1	20,486
3	CBF50-125	3	20,619
4	CBF50-123	3	22,039
5	CBF50-148	3	22,203
6	CBF50-118	3	22,701
7	CBF50-134	3	22,874
8	CBF50-109	3	22,932
9	CBF50-143	3	23,072
10	CBF50-044	1	23,098
11	CBF50-126	3	23,254
12	CBF50-116	3	23,613
13	CBF50-149	3	23,65
14	CBF50-138	3	23,687
15	CBF50-106	3	23,759
16	CBF50-135	3	24,8
17	CBF50-124	3	24,861
18	CBF50-128	3	25,081
19	CBF50-114	3	25,084
20	CBF50-027	1	25,27
21	CBF50-108	3	25,363
22	CBF50-113	3	25,367
23	CBF50-105	3	25,483
24	CBF50-006	1	25,512
25	CBF50-147	3	25,599
26	CBF50-101	3	26,055

FIGURE 1.1: Time series search engine.

to a certain partial aspect of the complete object.

1.2 Knowledge Discovery in Databases

Aside from simply retrieving similar objects, it is often of interest to discover hidden relations in a large amount of data. A similarity search can be considered as a 1:n algorithm, retrieving n objects for 1 query object. For large collections of data this approach is not feasible any longer. It is rather of interest to compare all objects to all other objects. To cope with such large datasets and to automatically detect potentially useful knowledge, algorithms for the so-called *knowledge discovery in databases* (KDD) have been proposed over the last years. According to [FPSS96], KDD is the non-trivial process of identifying valid, novel, potentially useful, and ulti-



Distance type: EDIST
Remaining calculations: 0

Distance type: EUCL
Remaining calculations: 0

Figure 1.2: Impact of different similarity measures on OPTICS clustering results.

mately understandable patterns in data. *Data mining* is described as a step within the KDD process. Data mining applies data analysis algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data. The other steps of the KDD process include data preprocessing and data reduction. Data mining however, can be considered as the core step of the KDD process, as the new knowledge is actually created during this step. Usually, data mining is based on a similarity measure. All techniques like clustering, classification, or the search for association rules depend on a notion of similarity. *Similar* objects are clustered together and classifiers try to describe classes of *similar* objects. In Figure 1.2 two OPTICS [ABKS99] clustering results are depicted. They differ only in the choice of the underlying similarity measure. Obviously the used similarity concept has a huge impact on any subsequent analysis step.

1.3 Temporal Data

In general, temporal data denotes a type of data where a temporal dimension is available aside from the data itself. This additional information is used to describe observations that vary over time. Aside from the detection of meaningful patterns in the actual recorded data, the temporal dimension allows for the search for temporal patterns as well. Such a pattern could be the correlation between the exposition to a chemical agent and a change in the gene expression rate. In this example, the point in time is an important attribute. Only when a change in the gene expression rate happens after the exposition to the agent, there might be a causal relationship. Temporal data consists of two main categories, bitemporal data and time series data.

- *Bitemporal Data* is the concept used in temporal databases. This term is used to summarize the two time concepts of temporal databases, in particular the *valid time* and the *transaction time*. The valid time denotes the time period during which a certain event is actually happening, or is *true*. In contrast to the valid time, the transaction time denotes the time during which a certain fact is stored in a database, even if it is not true any more. A prominent example is the administration of employees. A new employee may be inserted into the temporal database of a company several weeks before the employee actually begins to work for the company. As soon as the employee starts to work, the valid time begins. The database fact that the employee is working for the company is valid as long as the employee indeed works for the company. The transaction time however, may last several more years, as the company may be required by law to store facts about the former employee.
- *Time Series Data* describes more than just two states (true or false) for a certain observation. They are used to store measurements in a broad range of application domains. Think of biology where chemical concentrations within a cell may be measured for a number of points in time. Think of stock charts that reflect the price of a certain share for a certain moment in time. Two- or three-dimensional trajectories of moving objects are actually also time series. For a number of moments, the position of an object is recorded. This results in a two- or three-dimensional time series, respectively.

So, time series are the more complex representation for temporal data. Actually, bitemporal data can easily be converted to a time series, although usually a very simple-structured time series will result in this case. In this thesis we focus on time series, as they are the more general kind of temporal data. However, we use the notation *time series* and *temporal data* interchangeably.

1.4 Outline of the Thesis

This thesis is organized as follows:

Part I covers the preliminaries.

Chapter 1 describes the general idea of similarity search and its applications for knowledge discovery in databases.

Chapter 2 surveys related work in the area of similarity measures for time series objects. This includes the actual similarity measures as well as an overview of existing methods for the efficient calculation of similarity values, in particular by means of dimensionality reduction techniques. Furthermore we briefly review existing data mining techniques. Finally we introduce basic notations and definitions used in this thesis.

Chapter 3 describes the datasets we have used for the experimental comparisons throughout this work.

Part II introduces in total four new similarity measures for time series. In particular we introduce a threshold-based similarity measure, an amplitude-level-based similarity measure, an interval-focused similarity measure, and a similarity measure for uncertain time series.

Chapter 4 describes a new similarity measure based on thresholds. We consider two time series as similar, if they exceed a user-given threshold during similar time intervals. The exact values are not considered as they may be of no relevance for certain applications. This similarity measure is also of interest when amplitude values above a certain threshold level are error-prone, for example temperature sensors at very high temperatures. This chapter also presents an index structure which efficiently supports queries for the threshold-based similarity measure. This index structure allows for the specification of the threshold value at query time. We show how time series can be converted into a suitable representation and be stored in such a way, that queries using an arbitrary threshold value can be supported.

In *Chapter 5* we introduce the so-called amplitude-level-based similarity. This similarity measure considers a relevant amplitude range rather than a single threshold. This relevant range is scanned with a certain resolution and for each considered amplitude value features are extracted. We consider the change in the feature values over the amplitude values and thus, generate so-called feature sequences. Different features can finally be combined to answer amplitude-level-based similarity queries. In contrast to traditional approaches which aggregate global feature values along the time dimension, we capture local characteristics and monitor their change for different amplitude values. Furthermore, our method enables the user to specify a relevant range of amplitude values to be considered and so the similarity notion can be adapted to the current requirements.

Chapter 6 introduces the so-called interval-focused similarity measure. A user can specify one or several time intervals that should be considered for the calculation of the similarity value. Our main focus for this similarity measure was the efficient support of the corresponding query. In particular we try to avoid loading the complete time series objects into main memory, if only a relatively small portion of a time series is of interest. We propose a time series representation which can be used to calculate upper and lower distance bounds, so that only a few time series objects have to be completely loaded and refined. This representation is based on boxes that conservatively approximate a time series. The idea is to develop a sensible heuristic that generates suitable boxes. If too few boxes are generated, the time series can be only approximated very poorly and hence, the corresponding distance bounds are not very tight. Too many boxes however result in a large overhead for the storage of the boxes. Our method supports the specification of the relevant time focus at query time.

In *Chapter 7* we define a similarity measure for uncertain time series. This type of time series differs from standard time series. Uncertain time series consist of several amplitude values for each time slot. This can be due to multiple recordings or to errors in measurements, so that no exact value can be specified. In this situation, no exact distance value can be specified. Instead, a number of distance values can be observed comparing two uncertain time series. In fact, we show that this number of possible distance values is very high and so the need for an efficient distance estimation arises. We define two query types, the probabilistic bounded range query, and the probabilistic ranked range query. These queries return uncertain time series that fulfill the query-distance predicate with a given or the highest probability, respectively.

Part III focuses on the actual data mining step for temporal data.

Chapter 8 presents two approaches for the semi-supervised determination of threshold-values that are especially well suited for the calculation of threshold-based distance values for a given dataset and a given similarity concept. This concept can be specified by means of labeled training data. Our experiments showed that new knowledge can be generated, even if not all possible classes are available. This is especially important as this means, that partial knowledge about a certain dataset may be sufficient for the proposed algorithm. The so determined threshold can afterwards be used to discover further interesting relationships within the data.

In *Chapter 9* we present a data mining tool for time series data called *T-Time*. This tool can be used to perform classification or clustering analysis for time series. Furthermore it can be used to compare the impact of different similarity measures and of different dimensionality reduction techniques.

Part IV concludes this thesis.

In *Chapter 10* we summarize the contribution and findings of this thesis and explain possible directions for future work

This thesis is based on several publications. Table 1.1 gives an overview of these publications and maps them to the appropriate chapters of this thesis.

Topic	**Chapter**	**Publication**
Threshold-based similarity	4	[AKK$^+$06c] [AKK$^+$06b]
Amplitude-level-based similarity	5	[AKK$^+$08a]
Interval-focused similarity	6	[AKK$^+$07]
Semi-supervised threshold queries	8	[AKK$^+$06a] [AKK$^+$06d]
T-Time: A data mining tool for time series data	9	[AKK$^+$06e] [AKK$^+$08b]

Table 1.1: List of publications this thesis is based on.

Chapter 2

Similarity-Based Analysis of Temporal Data

In this chapter we give an overview of the field of similarity-based analysis of temporal data. At first we formally define the concept of time series.

2.1 Representing Temporal Data

A time series is a list of discrete values recorded at a certain point in time.

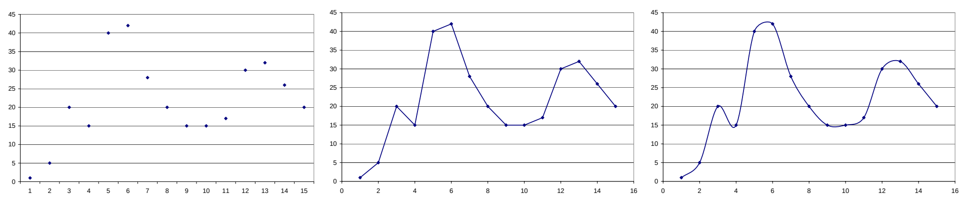
Definition 2.1 (Time Series).
A d-dimensional time series X is a sequence of tuples

$$\langle (x_1, t_1), .., (x_N, t_N) \rangle,$$

where $t_i \in T$ denotes a specific time slot and $x_i \in \mathbb{R}^d$ denotes the measurement recorded at time t_i. Furthermore, we assume that the sequence of tuples is ordered with respect to the time slots, i.e.

$$\forall i \in 1, .., N-1 : t_i < t_{i+1}$$

For the sake of presentation we assume one-dimensional time series throughout this work. However, the presented approaches can easily be extended to



(a) Discrete time series representation. (b) Linear interpolation. (c) Complex interpolation.

Figure 2.1: Interpolating discrete-valued time series.

cover the general d-dimensional case.

2.2 Interpolating Discrete-Valued Time Series

Usually, and as defined above, time series are discrete, i.e. they consist of single values at discrete points in time. This is due to the way, time series are usually created: by observations of a certain varying parameter. Obviously a real-world observation takes place at a discrete point in time. So in practice, each time series analysis has to cope with missing values. The standard approach to do so, is to linearly interpolate the existing values. Any two subsequent measurements or observations are used to define a linear function, so that afterwards a time series value can be calculated for any point in time, as long as it is not before the first recorded value, or after the last recorded value. A linearly interpolated time series is depicted in Figure 2.1(b) for the discrete time series presented in Figure 2.1(a). In case more information about the nature of the recorded data is available, other interpolation methods, for example based on polynomial functions, can be used as well (see Figure 2.1(c)).

Throughout this thesis we assume that missing time series values are available by means of linear interpolation.

Definition 2.2 (Alternative Notations for Time Series).
For the sake of presentation, we also use the notation $x(t)$ to denote the (potentially interpolated) time series value at time t. Furthermore, we may

omit the time information and simply consider a time series as a list of amplitude values. In this case we implicitly assume the distance in time between two time series values to be constant for all measurements. The actual points in time do not matter in this case. Let us assume the first value was recorded at $t = 1$, and the second value was recorded at $t = 2$. Then the list of values without temporal information actually corresponds to

$$X = \langle x_1, .., x_N \rangle = \langle (x_1, 1), (x_2, 2).., (x_N, N) \rangle,$$

Definition 2.3 (Length of a Time Series).
The length of a time series X is the number of tuples X consists of. The length is usually denoted by N. Note that this definition is consistent with linearly interpolated time series, if the length is defined as

$$N = (\text{LatestTimeSlot}) - (\text{EarliestTimeSlot}) + 1$$

as interpolating does not change the start or end point of a time series.

2.3 Similarity Measures for Complex Objects

In this section we cover the idea of similarity for complex objects in general. To definition of a similarity measure for complex objects is the first and probably most important step for all further analysis. Let us begin with a simple and general definition.

Definition 2.4 (Similarity Measure).
Let X and Y be two objects of a domain D. Then a similarity measure is a function $s : D \times D \to \mathbb{R}_0^+$ that assigns a similarity value to the pair X and Y. Usually higher similarity values are assigned to pairs of more similar objects. We also use similarity function *to denote a similarity measure.*

Usually the concept of *distance measures* or *distance functions* are used rather than similarity functions. The main difference is that lower values are assigned to more similar pairs of objects, as such objects have a low distance to each other in terms of similarity. A nice property of distance measures compared to similarity measures is that the value for identical objects is usually 0, i.e. $d(X, X) = 0$ for a distance function d. As 0 is the minimal value of the function range, no pair of non-identical objects can have a lower distance. On the other hand, when defining a similarity function, it is often not obvious what the maximal value for pairs of identical objects should be. Nevertheless, it is often possible to convert similarity values to distance values and vice versa. So, throughout this thesis we use the expressions *similarity function*, *similarity measure*, *distance measure*, and *distance function* interchangeably, whenever the actual meaning is obvious or irrelevant.

While for non-complex objects like real numbers a notion of similarity is easily agreed upon, for two complex objects like images, audio signals, three-dimensional objects, or time series, things unfortunately are not that clear. However, once such a similarity measure has been defined for complex objects, a vast range of possible applications and analysis approaches becomes feasible. These applications include information retrieval, search engines, classification of newly discovered instances of an complex object, clustering analysis, and many more. The problem of defining such a similarity measure is twofold: first, it is usually not obvious how similar two complex objects should be considered. Think of the three-dimensional objects taken from the Princeton Shape Benchmark [SMKF04] depicted depicted in Figure 2.2. How similar is a guitar to a collection of chairs and table? The answer obviously depends heavily on the person asking the question. A musician for example would probably think of a guitar as more similar to a piano as to some pieces of furniture. After all, it is rather difficult to produce music on a table with 4 chairs. On the contrary, a sales person querying a database of objects might think of the guitar as the outlier in the depicted group. Taking his point of view, the guitar is obviously much smaller and cheaper as the two other objects. This simple example underlines the need for different similarity functions for different application domains. One of the main aspects of this thesis is therefore to extend the existing notions of similarity, in particular those for time series.

The second difficulty involved in defining similarity measures is the cost for the actual computation of similarity or distance values. Images might

FIGURE 2.2: How to define similarity on complex objects?

have to be aligned by means of rotation before comparing them. Such an alignment step is even more costly for three-dimensional objects. Afterwards, complex mathematical functions might have to be evaluated to yield a single similarity value. So, for large-scale databases frequently involved in real-world problems, efficient access techniques have to be developed. Lower or upper bounds have to be sought after that allow for a faster yet less precise answer or that can be used as a filter step to get rid of a subset of candidates for the final result set. Consequently, efficiency considerations are the second main aspect of this thesis.

In the following, we will review existing similarity measures, as well as techniques to answer similarity queries efficiently.

2.4 Similarity Measures for Time Series

2.4.1 Minkowski Distance

The most prominent distance measure is the Euclidean distance which is a special case of the more general Minkowski distance for a parameter setting of $p = 2$

Definition 2.5 (Minkowski Distance).
Let X and Y be two time series of length N as defined above. Let $p \in \mathbb{R}, 1 \leq p \leq \infty$. Then $L_p(X,Y)$ is called the Minkowski distance of order p

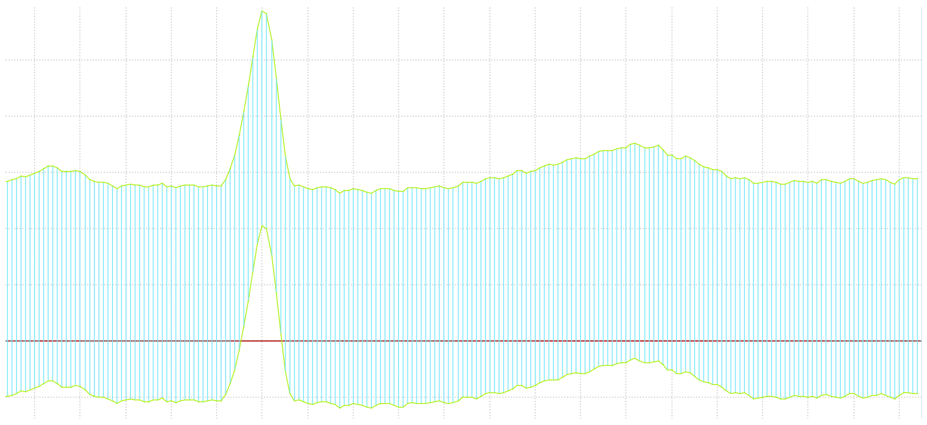


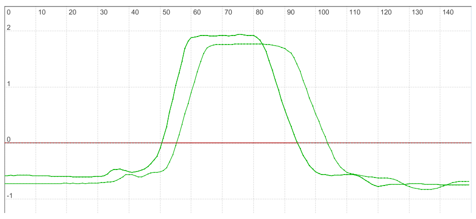
FIGURE 2.3: Computation of the Minkowski distance.

(or p-norm) between X and Y. L_p is defined as:

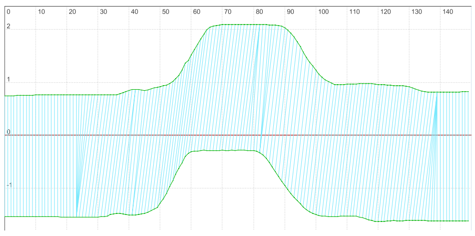
$$L_p(X,Y) = \sqrt[p]{\sum_{i=1}^{N} |x_i - y_i|^p}.$$

Further frequently used variants of the Minkowski distance include the Manhattan distance ($p = 1$) and the ∞-norm (Chebyshev distance). For $p \to \infty$ it can be shown that $L_p(X,Y) = \max_i(|x_i - y_i|)$. An example for the computation of the Minkowski distance is given in Figure 2.3. Each time slot of the first time series is compared to the corresponding time slot of the second time series. The closer the values at each time slot, the more similar the time series are considered. The only difference of the two depicted sample time series is an amplitude-wise shift. Otherwise they are very similar.

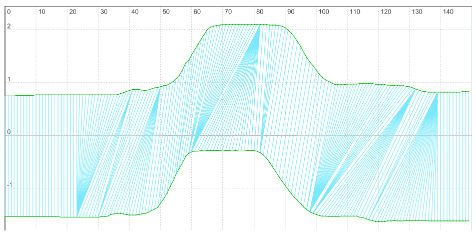
Especially the Euclidean distance is widely used and models the intuitive perception of similarity well for a broad range of applications. Furthermore it is efficiently to compute and thus is also suitable for large-scale applications. One of the main disadvantages is its sensitivity to even minor shifts in time.



(a) Time series to compare.



(b) DTW alignment with small warping distance.



(c) DTW alignment with large warping distance.

Figure 2.4: DTW alignment.

2.4.2 Dynamic Time Warping

The Dynamic Time Warping approach (DTW distance) was developed [BC94]. In [KCMP01] it was shown that DTW overcomes the problem of sensitivity to time shifts of the Minkowski distance. The DTW approach matches each time series value of the first time series to the best matching time series value of the second time series. This matching has to respect certain constraints, especially with respect to the distance in time of the matching partner, i.e. only a certain time warp is allowed. A certain time slot may be used several times as a matching partner, and so, time series of different length can be compared.

In Figure 2.4(a) two quite similar time series are depicted. Due to a small shift in time, standard Minkowski distances would yield relatively high distance values for this pair of time series. Figure 2.4(b) shows the matching the DTW yields for these time series. If the allowed time warp is increased (see Figure 2.4(c)), a different matching can be observed. In this case, the DTW distance will be even smaller, as more possible matching partners can be considered. However, in this case the computational cost will increase.

The DTW distance can be calculated by means of dynamic programming. The basic implementation is of quadratic complexity in the length of the time series. Depending on the warping window, i.e. the size of the allowed shift in time, it can be more efficiently calculated.

In [KP01] the authors introduced and enhanced version of the DTW, called Derivative Dynamic Time Warping.

2.4.3 Longest Common Subsequence Matching

A variant of the DTW distance is the family of distance values based on longest common subsequence (LCSS) matching between two time series. A problem of the DTW is that all time slots have to be matched. This is especially problematic in case of noisy data with large outlier values which can lead to low quality distance values. The LCSS addresses this problem by allowing gaps in the alignment of time series, i.e. not all time slots have to be matched. Examples for the use of LCSS based distance measures can be found in [DGM97] and in [VHGK03].

2.5 Efficient Handling of Temporal Data

As described above especially large collections of potentially long time series require efficient methods to access and process them. In the following we will review existing methods to deal with large amounts of time series data.

According to [FRM94] and to [KCMP01], the following properties are desired for an indexing method for time series.

- The query should be answered fast, in particular the computation time to answer a query should be faster than the sequential scan, which for large datasets is too slow.

- The method should be correct, i.e. all time series fulfilling the query predicate should be returned (no false dismissals). False hits are acceptable as they can be sorted out in a refinement step.

- The space overhead for the index structure should be small.
- The index structure should be able to store time series of different length and be able to answer queries of varying length.
- The index structure should be dynamic, i.e. it should be possible to delete, insert, or update time series.

The authors of [FRM94] proved that in order to guarantee no false dismissals, the following condition has to hold:

$$d_{index}(X, Y) \leq d_{exact}(X, Y)$$

for any two time series X and Y, where d_{index} is the distance in the index space, and d_{exact} is the actual and true distance between X and Y. This is known as the lower bounding lemma.

2.5.1 Vector Space Transformation

A straightforward approach to store time series data in an index structure is to consider time series of length n as a point in an n-dimensional vector space. In case of the Minkowski distance, the distance of two time series equals the Minkowski distance of the n-dimensional points in the vector space. This transformation allows for the storage in any multidimensional index structure like the R-tree [Gut84] or one of its many variants like the R*-tree [BKSS90]. Unfortunately, this approach suffers from the well-known *curse of dimensionality*, a term coined by [Bel61]. This effect describes the observation that the performance of index structures degrade very fast with increasing dimensionality of stored data. The reason for this is that the volume a vector space increases exponentially in the dimensionality. For uniformly distributed data, actually each data point is stored in its own leaf of the index structure. This phenomenon occurs for dimensionality values as low as 8 to 12 [CM99]. Obviously this value is too small for a broad range of

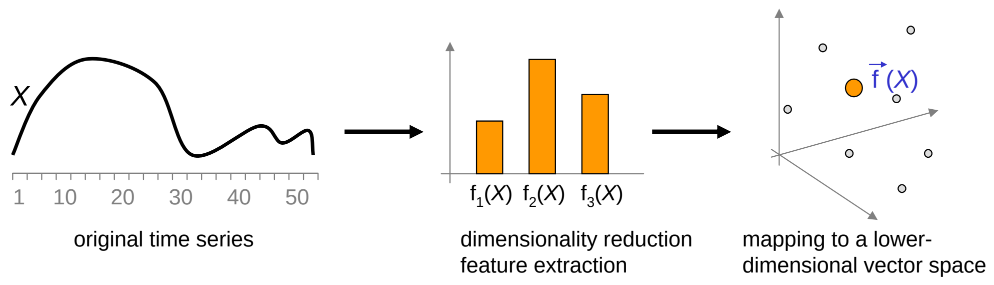


Figure 2.5: GEMINI approach.

real-world time series datasets whose entries frequently exhibit several thousand time slots. So, the work mentioned above [FRM94] introduced a general framework to overcome these problems.

2.5.2 Feature Space Transformation

The authors of [FRM94] introduced a general approach to index large amounts of multimedia data based on their observations described above. This approach is called *GEneric Multimedia INdexIng method* or *GEMINI*. This method is an instruction how to exploit any indexing structure fulfilling the lower bounding property. The general idea of the approach is to transform each input time series to be stored to a set of a few features. The number of features should be significantly less than the length of the time series. Instead of storing the time series, so called feature vectors are stored for each time series object, where the single feature value constitute the feature vector. An overview is given in Figure 2.5.

Based on the lower bounding property only those objects have to be refined whose index-based distance to a query object is below a given value. Refinement is the process of evaluating the true distance based on the exact time series representation. Typically, the refinement cost is much higher than the cost to calculate the index-based distance value. If the index-based distance is already higher than a given threshold, following the lower bounding lemma it is of no use to refine the corresponding object, as its true distance to a query object will be even higher or at least the same.

Techniques to extract such features are often called *dimensionality reduction techniques* and will be reviewed in the following.

2.5.3 Dimensionality Reduction Techniques

There exist a vast amount of dimensionality reduction techniques. Standard approaches include the Discrete Fourier Transform (DFT)[AFS93], extensions to the DFT [WFS04], the Singular Value Decomposition (SVD)[KJF97], the Discrete Wavelet Transform (DWT)[CF99], the Piecewise Aggregate Approximation (PAA) [YF00], the Adaptive Piecewise Constant Approximation (APCA) [KCMP01], and Chebyshev Polynomials [CN04].

The general idea of this techniques is to describe a time series with only a few coefficients in a way that allows to calculate lower distance bounds for the exact distance between two time series. This way, the time series can be stored in an index structure, avoiding the curse of dimensionality. The distance on the index structure can be used as a filter distance, so that afterwards only a few candidates have to be refined.

While these techniques are suitable for the indexing of Minkowski distances, only few methods have been proposed to index the DTW distance. In [KR02] the authors showed how it is possible to build an index structure for the DTW distance.

For further details on dimensionality reduction techniques, we refer the interested reader to the survey in [KCPM00].

2.5.4 Clipped Representations

In [RKBL05], a novel bit level approximation of time series for similarity search and clustering is proposed. Each value of the time series is represented by a bit. The bit is set to 1 if the value of the time represented by the bit is strictly above the mean value of the entire time series, otherwise it is set to 0. Then, a distance function is defined on this bit level representation that lower bounds the Euclidean distance and, by using a slight variant, lower

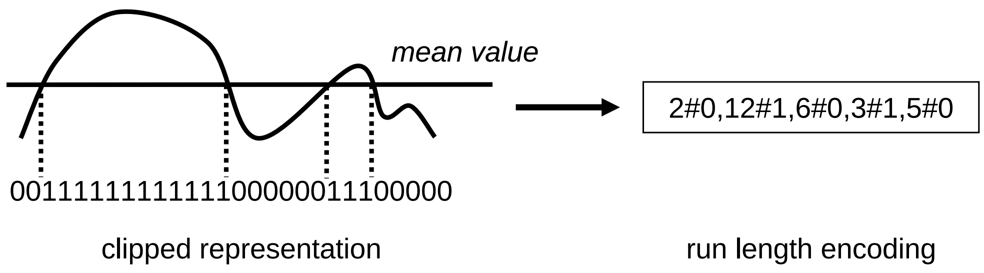


Figure 2.6: Clipped time series representation.

bounds DTW. An example for this representation is depicted in Figure 2.6. Note that the so derived bit sequences can be compressed, for example using run length encoding.

2.5.5 Filter-Refinement Architecture

The general approach to use an index structure in order to derive a first estimate for the exact distance, and to afterwards refine the remaining candidates is often called a filter-refinement architecture. In [KSF+96] the authors adapted the GEMINI approach to the k-Nearest-Neighbor search (cf. Definition 2.7). This idea was later improved in [SHP98] where the authors showed how to use an optimal filter-refinement approach with respect to the considered candidates. The key concept of the approach in [SHP98] is to dynamically update the filter criterion, whenever the exact distance of the k^{th} nearest neighbor retrieved so far is calculated. This distance can be used as the new filter distance. If an object which has not yet been refined has a lower bounding distance larger than the filter distance, it can not be an element of the resulting kNN set.

2.6 Data Mining on Time Series

Having defined a similarity measure on complex objects like time series, the two most basic queries for each more sophisticated data mining task, are the ε-range query and the k-nearest neighbor query. Together with efficient index structures, dimensionality reduction techniques, and suitable filter-refinement strategies, these query types lay the foundation for data mining techniques on large real-world datasets.

In the following sections we formally define these queries and give a brief overview of the many data mining approaches.

2.6.1 Query Types on Temporal Data

The first query type is the epsilon-range query.

Definition 2.6 (Epsilon-Range Queries).
Let $\mathcal{D}$ be the domain of complex objects. Let $d : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}_0^+$ be a distance function. The ε-range query *consists of a query object $q \in \mathcal{D}$ and a distance parameter $\varepsilon \in \mathbb{R}_0^+$. The* ε-range query *retrieves the set $Q_\varepsilon^{range}(q) \subseteq \mathcal{D}$ such that*

$$\forall x \in Q_\varepsilon^{range}(q) : d(q, x) \leq \varepsilon$$

The ε-range query can be used to answer a kNN query, which is defined as follows.

Definition 2.7 (k-Nearest-Neighbor Queries).
Let $\mathcal{D}$ be the domain of complex objects. Let $d : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}_0^+$ be a distance function. The k-nearest neighbor query *(kNN query) consists of a query object $q \in \mathcal{D}$ and a parameter $k \in \mathbb{N}^+$. The* kNN query *yields the smallest set $Q_k^{NN}(q) \subseteq \mathcal{D}$ that contains at least k elements such that*

$$\forall x \in Q_k^{NN}(q), \forall y \in \mathcal{D} \setminus Q_k^{NN}(q) :$$

$$d(q, x) < d(q, y)$$

A pseudocode for both query types can be found in the GEMINI paper [FRM94]. As mentioned above, an optimal kNN algorithm was introduced in [SHP98].

2.6.2 Data Mining Techniques

Clustering

Clustering tries to group objects into clusters. The idea of a cluster is that elements of the same cluster are more similar to each other than to elements of other clusters. So, the similarity is high within a cluster, and low across clusters. Following [Ber02], three of the main categories for clustering methods are hierarchical clustering,partitioning clustering, and density-based clustering. A further overview is given in [HK01].

Hierarchical clustering computes a cluster hierarchy, often represented as a dendrogram. Some clusters contain further child clusters. A hierarchical clustering structure is usually either obtained by iteratively splitting of the dataset, or by merging smaller clusters to a parent cluster. The first approach is called *divisive* and starts with only one cluster containing the complete data. This cluster is recursively split, until some stop criterion is fulfilled. *Agglomerative* approaches start with clusters consisting of only one object. In each iteration, clusters are merged, until all the data is contained in the root cluster. Prominent examples of hierarchical clustering methods include Single Link [Sib73], CURE [GRS98], and BIRCH [ZRL96].

Partitioning clustering splits the available data into disjoint clusters. One of the first clustering algorithms of this category was the k-means algorithm [Mac67]. Further examples are the k-medoid-based approaches PAM and CLARA [KR90], and CLARANS [NH94].

Density-based clustering groups objects into clusters according to a density criterion. While approaches like k-means are often restricted to the creation of convex clusters, density-based approaches usually detect clusters of any shape. DBSCAN [EKSX96] and its hierarchical variant OPTICS [ABKS99] as well as DENCLU [HK98] are prominent examples in this category.

Classification

Classification is a supervised data mining task, i.e. a set of labeled training data is available based on which a model can be learned. This model (the classifier) can afterwards be used to predict the class of a newly discovered object from the same domain as the training data. Important categories of existing classification techniques include decision trees, statistical methods, instance-based learners, and Support Vector Machines [Kot07].

Decision trees consist of nodes representing features of the instances to be classified. Each value the feature can assume is represented by a branch leading to the next node or to a leaf in case the object has successfully be classified. The most well-known decision tree algorithm is the C4.5 algorithm [Qui93]

Statistical methods assign probability values for the correct rather than a single class label. Naive Bayesian networks are relatively simple classifiers with independence assumptions concerning the values of the different features of an object. However they were shown to be quite competitive in [DP97]. The more general Bayesian networks or belief networks are able to model probability relationships between a set of features. However they are quite difficult to compute [Kot07].

Instance-based learners are also called lazy learners, as they do not derive an explicit model like a decision tree or a Bayesian network for a given training set. They rather use the training set each time a classification task is to be performed. The most well-known instance-based learner is the k-nearest neighbor classifier [CH67] with its many variants.

Support Vector Machines are one of the newest classification approaches and were introduce in [Vap95]. SVMs try to separate two classes with a hyperplane that maximizes the so called margin, i.e. the distance to both classes. This way, overfitting is minimized. Often, two classes are not linearly separable by a hyperplane. In this case the original data points are mapped to a higher-dimensional Hilbert space H. During the training phase, dot products in H have to be evaluated. If there exists a so called kernel function which can be evaluated directly in the original feature space, but whose result equals the dot product in H, it is not necessary to explicitly map all training instances to H. This is known as the “kernel trick” [SBS99].

2.7 Basic Notations

In this section we summarize basic notations used throughout this thesis. If required, more specific notations will be introduced in the corresponding sections.

- i, j, k for integers
- $\mathcal{D}$: the domain of time series objects or a database containing time series objects
- N, n: length of a time series
- T: the time domain
- τ: a threshold value
- Q: a query time series
- X, Y: the time series X and Y
- $TQ_{\varepsilon}^{range}(Q, \tau)$: Threshold-based ε-range query
- $TQ_{k}^{NN}(Q, \tau)$: Threshold-based k-nearest neighbor query
- $IQ_{\varepsilon}^{range}(Q, \mathcal{I})$:Interval-focused ε-range query
- $IQ_{k}^{NN}(Q, \mathcal{I})$: Interval-focused k-nearest neighbor query
- $PQ_{\varepsilon}^{range}(\mathcal{Q}, \tau)$: Probabilistic Bounded Range Query (PBRQ)

- $PQ_{\varepsilon}^{rank}(\mathcal{Q}, \tau)$: Probabilistic Ranked Range Query (PRRQ)

Chapter 3

Datasets used in the Experimental Evaluation

In this chapter we describe the datasets used in the experimental sections of this thesis. A brief overview is given in Table 3.1.

3.1 Audio Dataset

In order to create a set of similar datasets with varying size and varying length of the time series, we created several audio datasets. These dataset contain time sequences, expressing the temporal behavior of the energy and frequency in music sequences. The exact length of the time series and the size of the used dataset is specified whenever one of these datasets is used. Example time series are depicted in Figure 3.1. This dataset was mainly used for efficiency related experiments.

3.2 Air Pollution Dataset

The data on environmental air pollution was provided by the Bavarian State Office for Environmental Protection, Augsburg [LFU], and contains the daily

Dataset	# time series	length	# classes
Audio	up to 700000	up to 10000	-
Air Pollution	up to 1800	48	varying
Gene Expression	6000	8/16	varying
GunX	200	150	2
Trace	200	275	4
CBF	150	127	3
SynCtrl	600	6000	6
Leaf	442	150	6

Table 3.1: Summary of test datasets.

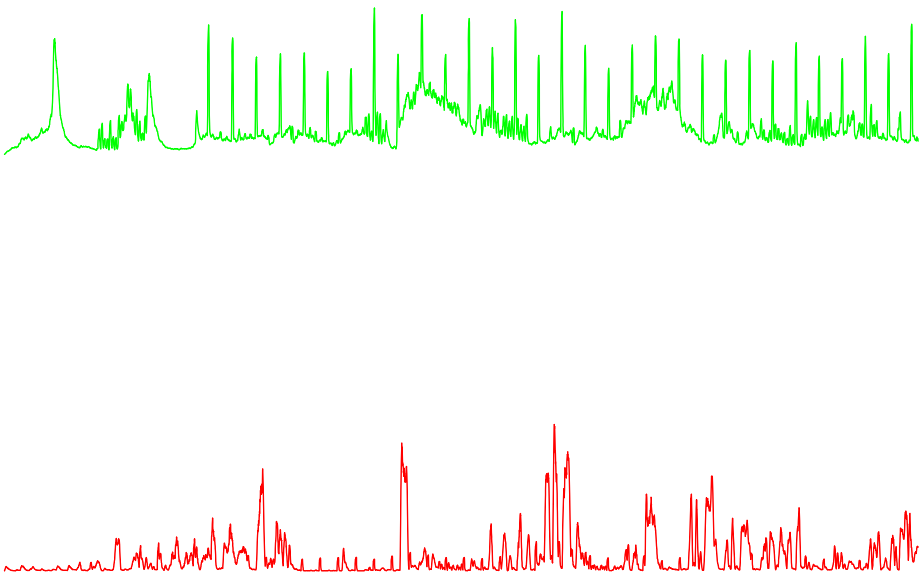


Figure 3.1: Example time series of audio dataset.

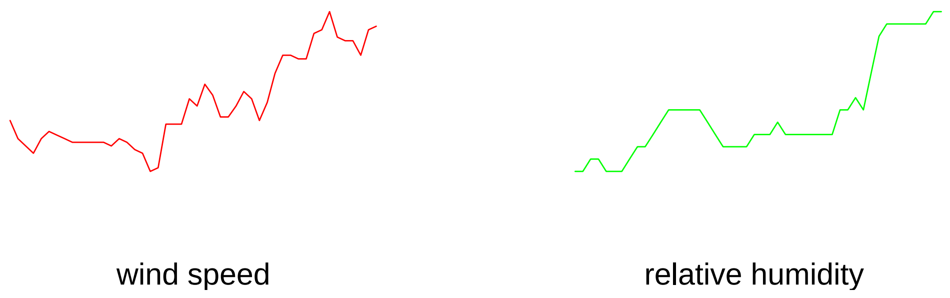


FIGURE 3.2: Example time series of the environmental air pollution dataset.

measurements of 8 sensor stations distributed in and around the city of Munich for the years 2000 to 2004. One time series represents the measurements of one station at a given day containing 48 values for one of 10 different parameters like temperature or ozone concentration. Example time series are presented in Figure 3.2.

3.3 Gene Expression Dataset

The gene expression datasets contain the expression levels of approximately 6,000 genes of the yeast Saccharomyces cerevisiae, which is commonly known as baker's yeast [SSZ$^+$98]. The first subset is the Gene Expression Omnibus [BTW$^+$06, EDL02] dataset GDS 38 and contains 2562 entries. Gene expression levels were recorded every 7 minutes during the cell cycle. In total 16 measurements were recorded. The second subset is the GDS 30 set consisting of 2628 entries. 8 measurements over 90 minutes were recorded, after the cells were exposed to the chemical agent diamide. In order to obtain a reference classification for our experiments, we used the hierarchical classification system of the Gene Ontology project [ABB$^+$00]. The number of the classes varied according to the selected classification level. Example time series for both subsets are depicted in Figure 3.3.

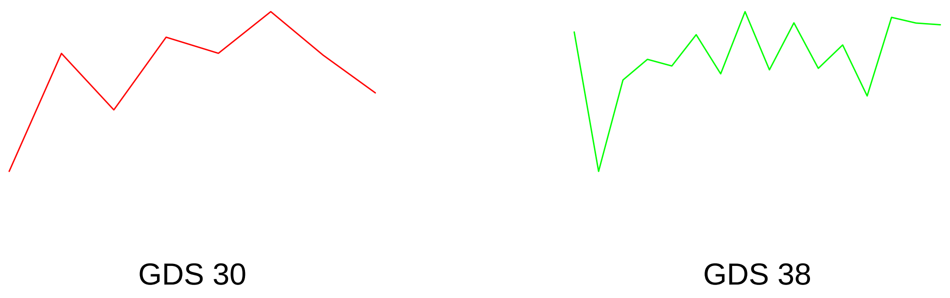


Figure 3.3: Example time series of the Gene Expression dataset.

3.4 Standard Datasets

The standard datasets are derived from diverse fields and cover the complete spectrum of stationary/non-stationary, noisy/smooth, cyclical/non-cyclical, symmetric/asymmetric etc. data characteristics. They are available from the UCR Time Series Data Mining Archive [KF02]. Due to their variety, they are often used as benchmark for novel approaches in the field of similarity search in time series databases. We selected four datasets of this repository, in particular the GunX dataset, the Trace dataset, the CBF dataset, and the SynCtrl dataset. We cover these datasets in the following sections.

3.4.1 GunX

This dataset is a two-class dataset from the field of video surveillance. The dataset contains two classes, each consisting of 100 instances. All instances were created by using a female a male actor which had to perform two different tasks corresponding to the two classes. The first class is called "Gun-Draw". For this class, the actors had their hands by their sides. Then they drew a gun from a hip-mounted holster, point it at a target for approximately one second and returned the gun to the holster. The second class, "Point", corresponds to a similar movement, although the index finger is used to aim at a target instead of a real gun. For both classes, the centroid of the right hand in X-axes was tracked. Each instance has the same length of 150 data points. Example time series for both classes are depicted in Figure 3.4.

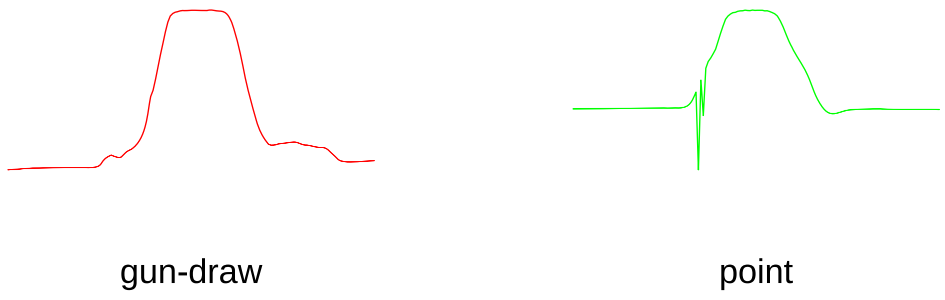


Figure 3.4: Example time series of the GunX dataset.

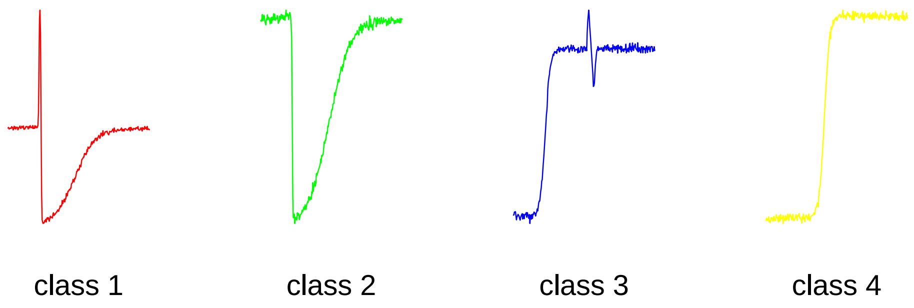


Figure 3.5: Example time series of the Trace dataset.

3.4.2 Trace

The trace dataset is a four-class dataset which is a subset of the Transient Classification Benchmark (trace project) used in [Rov02] for plant diagnostics. It is a synthetic dataset designed to simulate instrumentation failures in a nuclear power plant. The full dataset consists of 16 classes, 50 instances in each class. Each instance has 4 features. The *Trace* subset only uses the second feature of class 2, and the third feature of class 3 and 7. Hence, this dataset contains 200 instances, 50 for each class. All instances are linearly interpolated to have the same length of 275 data points. Examples can be found in Figure 3.5.

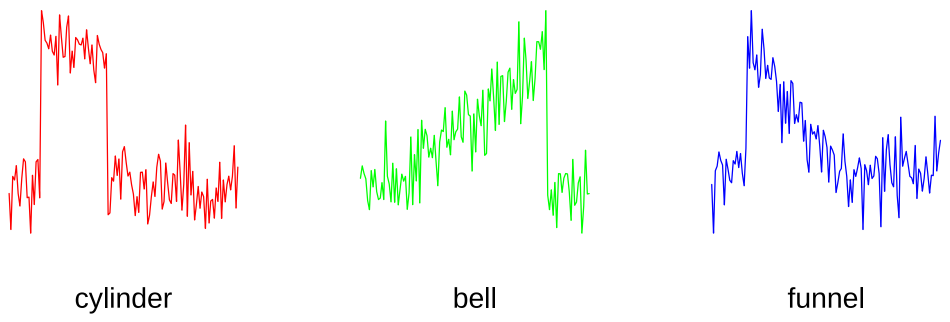


FIGURE 3.6: Example time series of the CBF dataset.

3.4.3 CBF

The CBF dataset is derived from the artificial cylinder-bell-funnel task, originally proposed by in [Sai94]. The task is to classify a time series as one of the three classes, cylinder, bell, or funnel. We used a subset (50 time series from each class) of the original dataset, containing 100 cylinders, 100 bells and 100 funnels. Example time series are depicted in Figure 3.6.

3.4.4 SynCtrl

The SynCtrl dataset contains 600 examples of control charts synthetically generated by a process described in os [AM99]. This dataset consists of the Cyclic pattern subset of the control chart data from the UCI KDD archive (kdd.ics.uci.edu). The data is effectively a sine wave with noise consisting of 6,000 data points. There are six different classes, each class consisting of 100 instances. A member of each class is depicted in Figure 3.7.

3.4.5 Leaf

This dataset is based on an image dataset of leafs. According to [RK05] these leaf images were transformed to time series by measuring the distance of leaf contour points to the leaf centroids. Example time series for the 6 different classes can be found in Figure 3.8.

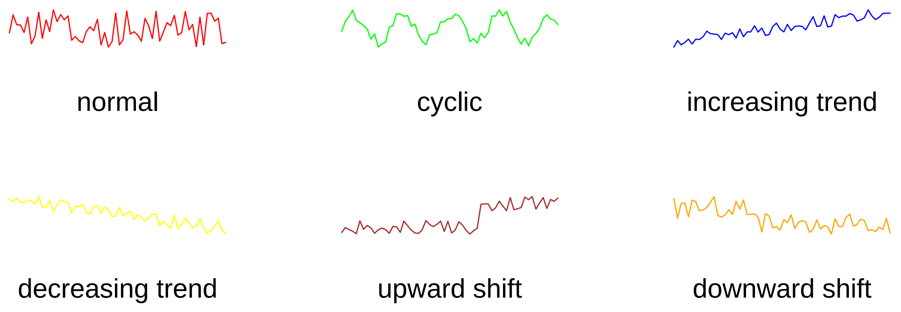


FIGURE 3.7: Example time series of the SynCtrl dataset.

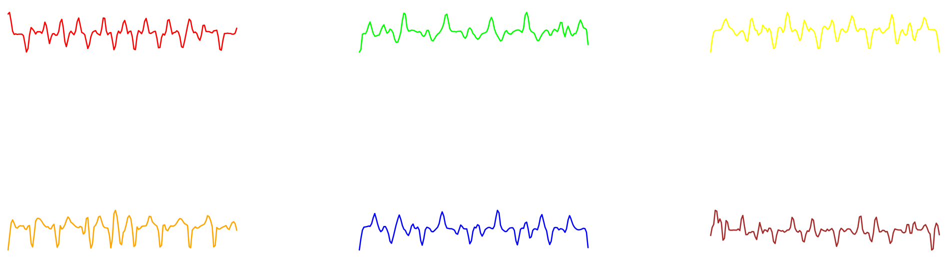


FIGURE 3.8: Example time series of the Leaf dataset.

Part II

New Similarity Measures for Time Series Data

Chapter 4

Threshold-Based Similarity

Time series are quite often very long, consisting of thousands of values. So, the comparison of two such long time series can be very expensive, especially when the exact values over the complete course of time is considered. However, there are a lot of data mining applications where such exact information is not required or even worse, is not desired. Often, it is more sensible to compare two time series qualitatively. This can be the case if a certain threshold is of interest, for example a legal threshold of some chemical agent. Another possibility is that above a certain amplitude level, the measurements are more prone to errors, and so the exact measurement is of no interest.

In this chapter, we introduce a novel type of similarity measure for time series, called *threshold similarity*. The corresponding similarity query is called *threshold query*. A threshold query consists of a query time series Q and a threshold vale $\tau \in \mathbb{R}$. The database time series as well as the query sequence Q are decomposed into time intervals of subsequent elements where the values are (strictly) above τ. Now, the threshold query returns the time series objects of the database which have a similar interval sequence of values above τ. Note, that the complete set of exact amplitude values are irrelevant for the query. The time intervals of a time series X only indicate that the amplitude values of X corresponding to the time intervals are above the given threshold τ. The concept of threshold queries can be useful in many practical application domains.

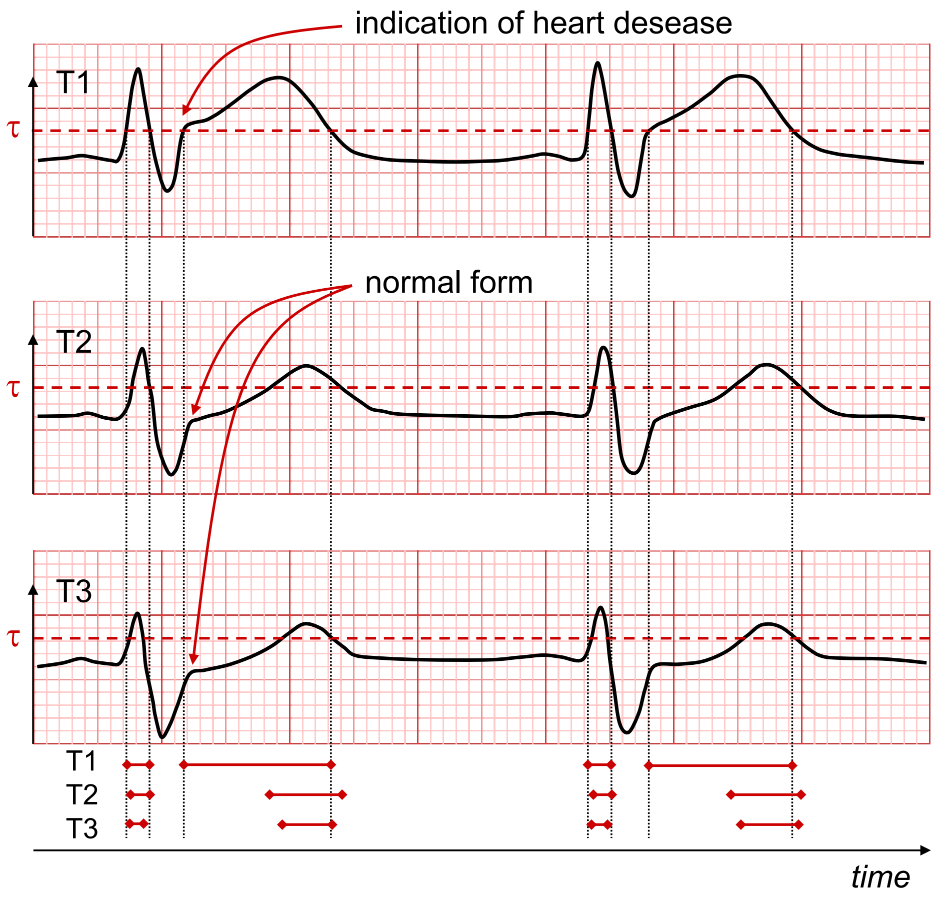
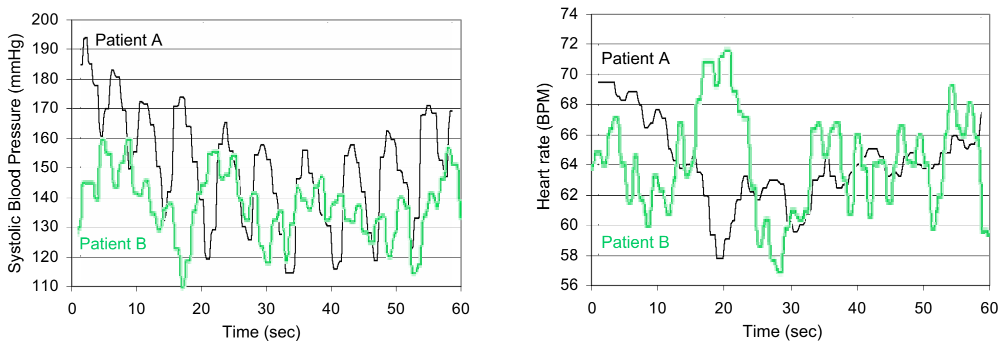


FIGURE 4.1: Sample application of threshold-based similarity.

A sample application from *medical analysis* is depicted in Figure 4.1 where three real electrocardiogram (ECG) plots $T1$, $T2$, and $T3$ are shown. Plot $T1$ indicates a high risk for cardiac infarct due to the abnormal deflection after the systole (ST-T-phase), whereas $T2$ and $T3$ both show a normal curve indicating a low risk. For the examination of time series with respect to this abnormal characteristic, there is no need to examine the entire curve. A better way to detect such kind of characteristics is to analyze only the relevant parts of the time series, for instance observing those parts of the time series which exceed a specified threshold as depicted in our example. Let us now consider the time interval sequences (below the ECG-curves) which correspond to the time frames within which the time series exceed the threshold τ. We can observe that the time interval sequences derived



(a) heart rate (b) systolic blood pressure

FIGURE 4.2: Heart rate and systolic blood pressure after drug treatment.

from $T2$ and $T3$ differs marginally. In contrast, the time series $T1$ exhibits a different characteristic, caused by the ECG-aberration which indicates the heart disease.

For the *pharmaceutical industry* it is of interest which drugs have a similar impact on patients at a similar time relative to the exposition to a certain drug. Obviously, effects such as a certain blood parameter exceeding a critical level τ are of particular interest. Figure 4.2 depicts heart rate and blood pressure measurements for two patients. The response of patient A differs significantly from that of patient B. Threshold queries can help to identify patients with a similar pattern even if the exact values are not too similar. The exact values may be influenced by the personal disposition of different persons or even by different methods for measuring the observed parameter.

The analysis of *environmental air pollution* becomes more and more important and has been performed by many European research projects in the recent years. The amount of time series data derived from environmental observation centers, increases drastically with elapsed time. Furthermore, modern sensor stations record many attributes of the observed location simultaneously. For example, German state offices for environmental protection maintain about 127 million time series each representing the daily course of several air pollution parameters. An effective and efficient processing of queries like "return all ozone measurements which exceed the threshold

$\tau_1 = 50\mu g/m^3$ at a similar time as the temperature reaches the threshold $\tau_2 = 25°C$" can be very useful. Obviously, the increasing amount of data to be analyzed poses a big challenge for methods supporting threshold queries efficiently.

In *molecular biology* the analysis of gene expression data is important for understanding gene regulation and cellular mechanisms. Gene expression data contains the expression level of thousands of genes, indicating how active a gene is over a certain time frame. The expression level of a gene can be up (indicated by a positive value) or down (negative value). From a biologist's point of view, it is interesting to find genes that have a similar up and down pattern because this indicates a functional relationship among the particular genes. Since the absolute up/down-value is irrelevant, this problem can be represented by a threshold query. Each gene expression sequence is converted to an interval sequence, indicating the time slots of the gene being in an up-regulated state. Genes with a similar interval sequence thus have a similar up and down pattern.

In summary, our contributions are the following:

- We introduce and formalize the novel concept of threshold-based similarity for time series databases.
- We present a novel data representation of time series which support threshold queries efficiently.
- We introduce an efficient algorithm for threshold queries based on the new time series representation.

The remainder of this chapter is organized as follows. Section 4.3 formally introduces the notion of threshold queries. In Section 4.4, we show how time series can be represented in order to support threshold queries for arbitrary threshold values efficiently. Section 4.5 describes efficient query algorithms based on the proposed representation. The effectiveness and efficiency of our algorithms are evaluated in Section 4.6.

4.1 General Idea

At first, we will describe the general concept of threshold-based similarity search. Let$X$ and Y be time series and let τ be an amplitude threshold. X and Y are considered similar if their amplitudes exceed the threshold τ within similar time intervals. Using threshold similarity, the exact values of the time series are not considered. Rather, it is only examined whether the time series at similar time intervals are above or below the given threshold τ. Thus, time series can be considered as similar, even if their absolute values are considerably different, as long as they have similar time frames during which the time series exceeds the specified query threshold τ. Then, the processing of queries like"retrieve all pairs of sequences of ozone concentration which are above the critical threshold of $50\mu g/m^3$ at a similar time" is reduced to comparing sequences of time intervals. Usually, the number of intervals is much smaller than the number of exact values per time series and can be organized more efficiently. If the aggregated threshold-based representation in form of time intervals for each time series is given in advance, it is obvious that the threshold queries can be answered more efficiently compared to the situation where the time intervals are not given in advance.

4.2 Related Work

As described in Chapter 2, time series can indexed by spatial access methods such as the R-tree and its variants [Gut84]. However, most spatial access methods degrade rapidly with increasing data dimensionality and so, dimensionality reduction techniques like DFT [AFS93], DWT [CF99], PAA [YF00], SVD [KJF97],APCA [KCMP01], or Chebyshev Polynomials [CN04] are used to efficiently index time series as described in [FRM94].

However, all techniques which are based on dimensionality reduction cannot be applied to threshold queries because necessary temporal information is lost. Usually, in a reduced feature space, the required intervals indicating that the time series is above a given threshold cannot be generated. In addition, the approximation generated by dimensionality reduction techniques cannot be used for our purposes because they still represent the exact course of the time series rather than intervals of values above a threshold.

For a lot of applications, the Euclidean distance is too sensitive to minor distortions along the time axis. It has been shown that Dynamic Time Warping (DTW) can fix this problem [KCMP01]. However, DTW is not applicable to threshold queries because it considers the absolute values of the time series rather than the intervals of values above a given threshold.

The bit level approximation introduced in [RKBL05] is restricted to a certain predetermined threshold and so, this approach is not applicable for threshold queries where the threshold is not known until query time.

To the best of our knowledge, there does neither exist any access method for time series, nor any similarity search technique which efficiently supports threshold queries.

4.3 Threshold-Based Queries

In this section, we formally introduce the concept of threshold-based queries or threshold queries.

4.3.1 Threshold-Crossing Time Intervals

At first we show how a given threshold value τ is used to define a collection of intervals, the so called Threshold-Crossing Time Interval Sequence.

Definition 4.1 (Threshold-Crossing Time Interval Sequence).
Let X be a time series as defined in Definition 2.1 of length N, let $\tau \in \mathbb{R}$, and let T be the time domain. Then the threshold-crossing time interval sequence *of X with respect to τ is a sequence $S_{\tau,X} = \langle (l_j, u_j) \in T \times T : j \in \{1, .., M\}, M \leq N \rangle$ of time intervals, such that*

$$\forall t \in T : (\exists j \in \{1, .., M\} : l_j < t < u_j) \Leftrightarrow x(t) > \tau.$$

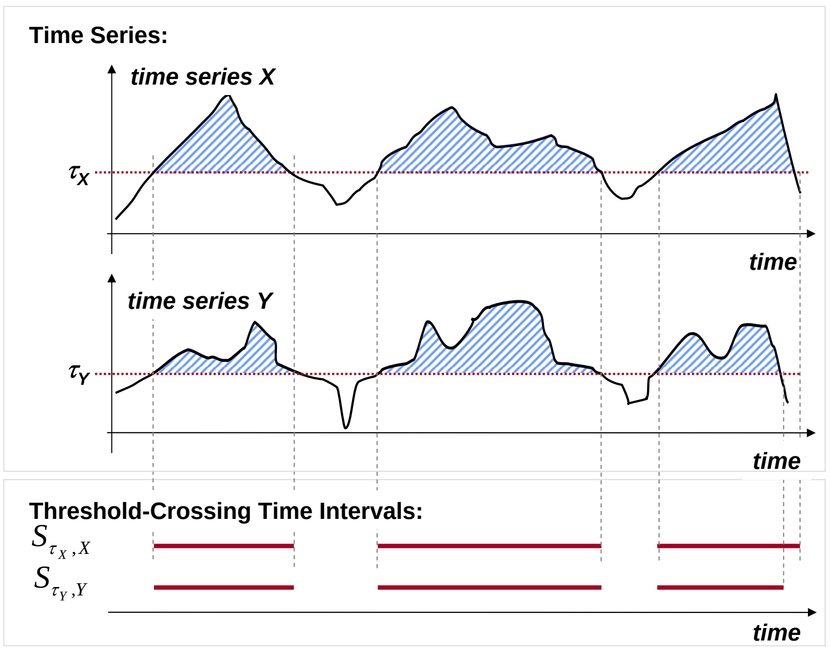


FIGURE 4.3: Threshold-Crossing Time Intervals

The value τ *is called the* threshold.

Note that we omit the τ parameter in $S_{\tau,X}$ if the choice for τ is obvious or no specific value for the threshold is given.

The example shown in Figure 4.3 depicts two threshold-crossing time interval sequences for two time series, X and Y with respect to two different threshold values τ_X and τ_Y.

4.3.2 Similarity Model for Threshold-Crossing Time Interval Sequences

In order to define a threshold-based similarity function on time series we have to define a similarity function on the interval sequence representations derived for a certain threshold. The threshold-crossing time interval sequences consist of single intervals, so the first step is to define a similarity function on single intervals. This function can afterwards be used to calculate a similarity value for a sequence of intervals.

We consider intervals to be similar to each other, if they have similar start points and similar end points.

Definition 4.2 (Distance between Time Intervals).
Let $t1 = (t1_l, t1_u) \in T \times T$ and $t2 = (t2_l, t2_u) \in T \times T$ be two time intervals. Then the distance function $d_{int} : (T \times T) \times (T \times T) \rightarrow \mathbb{R}$ between two time intervals is defined as:

$$d_{int}(t1, t2) = \sqrt{(t1_l - t2_l)^2 + (t1_u - t2_u)^2}$$

Let us note, that intervals correspond to points in a two-dimensional space, where the starting point corresponds to the first dimension and the ending point corresponds to the second dimension. This transformation is explained in more detail in the next section (cf. Section 6.3). Then the above definition of a distance function on intervals corresponds to the Euclidean distance in this two-dimensional space. While it is also possible to use other Minkowski metrics, we only use the Euclidean distance throughout this thesis. As we will show in the experimental section 4.6, the differences between different Minkowski metrics are negligible.

Since for a certain threshold τ a time series object is represented by a sequence of time intervals, we need a distance measure for sequences of intervals. As these intervals are naturally ordered by their starting points and as the intervals do not overlap each other, we can consider the threshold-crossing time interval sequences as sets of intervals without loss of generality. Several distance measures for sets have been introduced in the literature [EM97]. We use the Sum of Minimum Distances (SMD). Let S_1 and S_2 be two sets. The idea of the SMD is as follows: at first, each element of S_1 is matched to the best suited element in S_2 and afterwards the same is done for each element of S_2. The process of matching two element is based on a distance function defined on two elements of the sets.

In our case, when given two time series, each threshold-crossing time interval of the first time series will be mapped to its most similar counterpart of the second time series. Obviously two threshold-crossing time interval sequences do not necessarily have the same cardinality, so we follow [KM04] and adapt the original definition of the *SMD* to our needs, by normalizing the distance value by the cardinalities of the interval sets. Finally we are able to define the *threshold-distance.*

Definition 4.3 (Threshold-Distance).
Let X and Y be two time series and S_X and S_Y be the corresponding threshold-crossing time interval sequences. Then the threshold distance d_{TS} is defined as

$$d_{TS}(S_X, S_Y) = \frac{1}{2} \cdot \left(\frac{1}{|S_X|} \cdot \sum_{s \in S_X} \min_{t \in S_Y} d_{int}(s,t) + \frac{1}{|S_Y|} \cdot \sum_{t \in S_Y} \min_{s \in S_X} d_{int}(t,s) \right)$$

For the sake of clarity, in the above definition we assumed both interval sequences were created using the same threshold. However, this is not a necessary constraint. As already mentioned, the idea of this distance function is to map every interval from one sequence to the closest (most similar) interval of the other sequence and vice versa. This distance measure has a further advantage. Time series having similar shapes, i.e. showing a similar behavior, may be transformed into threshold-crossing time interval sequences of different cardinalities. Since the above distance measure does not consider the cardinalities of the interval sequences, this distance measure is quite suitable for time interval sequences. Another advantage is that the distance measure mainly considers local similarity. This means, that for each time interval only its nearest neighbor (i.e. closest point) of the other sequence is taken into account. Other intervals of the counterpart sequence have no influence on the result.

4.3.3 Similarity Queries based on Threshold Similarity

Based on the new distance measure introduced in the last sections, we can now extend the two most widely used similarity queries, the *distance range query* and the *k-nearest-neighbor query.* As specified in Definition 2.6, the distance range query retrieves all objects of a database whose distance to a given query object Q is smaller or equal to a given distance value ε. This query type is also called *ε-range query*. The *k-nearest neighbor query* (kNN query) (cf. Definition 2.7) reports the k most similar objects to Q for a given value of k.

Definition 4.4 (Threshold-Based ε-Range Query).
Let $\mathcal{D}$ be a set of time series objects. The threshold-based ε-range query *consists of a query time series Q, a query threshold $\tau \in \mathbb{R}$, and a distance parameter $\varepsilon \in \mathbb{R}_0^+$. The* threshold-based ε-range query *retrieves the set $TQ_\varepsilon^{range}(Q,\tau) \subseteq \mathcal{D}$ such that*

$$\forall X \in TQ_\varepsilon^{range}(Q,\tau) : d_{TS}(S_Q, S_X) \leq \varepsilon$$

Analogously we extend the definition of the kNN query as follows.

Definition 4.5 (Threshold-Based k-Nearest-Neighbor Query).
Let $\mathcal{D}$ be a set of time series objects. The threshold-based k-nearest neighbor query *consists of a query time series Q, a query threshold $\tau \in \mathbb{R}$, and a parameter $k \in \mathbb{N}^+$. The* threshold-based k-nearest neighbor query *yields the smallest set $TQ_k^{NN}(Q,\tau) \subseteq \mathcal{D}$ that contains at least k elements such that*

$$\forall X \in TQ_k^{NN}(Q,\tau), \forall Y \in \mathcal{D} \setminus TQ_k^{NN}(Q,\tau) :$$
$$d_{TS}(S_Q, S_X) < d_{TS}(S_Q, S_Y)$$

Again, this definition could be adapted to different threshold values for different time series. However, as the query time series Q usually is of the same application domain as the collection of time series the query is executed on, the standard approach is to use the same threshold value for all time series. In the following we will also refer to both query types as "threshold query" if it is not necessary to distinguish between the two different query types. We use the abbreviation $TQ(Q,\tau)$ to denote this generalized query type.

4.4 Index Support for Threshold Queries

A straightforward approach to execute a threshold query $TQ(Q,\tau)$ is to sequentially read each time series X from the database. Afterwards we compute the threshold-crossing time interval sequence $S_{\tau,X}$ for each time series for the given threshold which allows us to compute the threshold-similarity function $d_{TS}(S_{\tau,Q}, S_{\tau,X})$). Finally, we report the time series which fulfill the query predicate according to Definitions 4.4 and 4.5. However, if the time series database contains a large number of objects and the time series are reasonably large, then obviously this way of performing the query becomes unacceptably expensive. So, in this section, we present an access method which efficiently supports threshold queries. In particular, we show how to efficiently store and access all the required threshold-crossing time interval sequences.

We present two approaches for the management of time series data. The key point of both approaches is that we do not need to access the complete time series data at query time. Instead, only partial information of the time series objects is required. At query time we only need the information at which time frames the time series is above the specified threshold. We can save a lot of IO cost if we only access the relevant parts of the time series at query time. The basic idea of our approach is to pre-compute the threshold-crossing time interval sequences for each time series object and store it in such a way it can be accessed efficiently.

For the sake of clarity, we first present a basic approach where the threshold value τ is known in advance. Afterwards, we present the general approach which supports an arbitrary choice of τ at query time.

4.4.1 Threshold-Based Indexing for a Fixed Threshold

Let us assume that the query threshold τ is fixed for all queries. Then we can compute the corresponding threshold-crossing time interval sequence $S_{\tau,X}$ for each time series $X \in \mathcal{D}$. Consequently, each time series object is represented by a sequence of intervals. There are several methods to store intervals efficiently, e.g. the RI-Tree [KPS01b]. However, these index structure are usually designed to support intersection queries on interval data. Our approach however, requires the support of similarity queries according to the

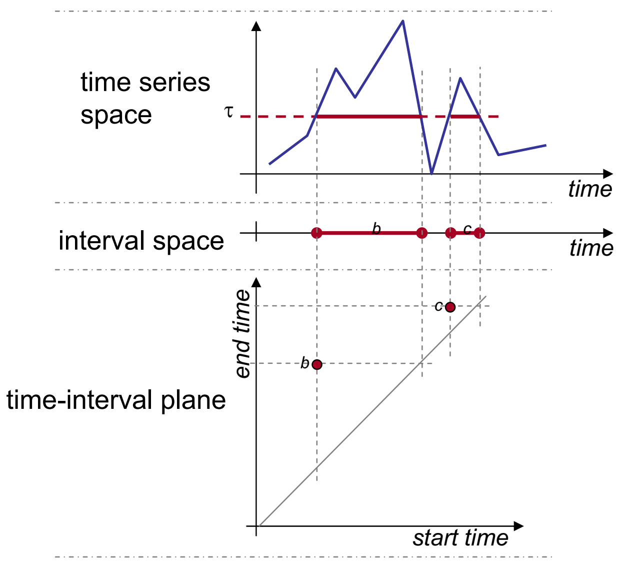


Figure 4.4: Mapping of time intervals to the time-interval plane.

similarity model for threshold-crossing intervals (cf. Definition 4.2) and the similarity model based on the SMD (cf. Definition 4.3). Furthermore, it is not possible to use these index structures for the general case where τ is not known in advance. Therefore we propose a solution which supports similarity queries on intervals and which can be easily extended to support queries with arbitrary τ.

Time intervals can be considered as points in a two-dimensional plane [GG84]. In the following, we will refer to this plane as the time-interval plane. The one-dimensional intervals (*native space*) are mapped to the time-interval plane by using their start and end points as two-dimensional coordinates. This representation has several advantages for the efficient management of intervals.

- The position of large intervals, which are located in the upper-left region, substantially differs from the position of small intervals (located near the diagonal) .
- The most important advantage is that the Euclidean distance in this plane corresponds to the distance function of intervals according to

Definition 4.2.

The complete threshold-crossing time interval sequence is represented by a set of 2-dimensional points in the time-interval plane. The transformation from the original time series to the point set in the time-interval plane is depicted in Figure 4.4.

In order to efficiently manage the point sets of all time series objects, we can use a spatial index structure like the R*-tree [BKSS90]. In particular, the R*-tree is very suitable for managing points in low-dimensional spaces, where the points are not equally distributed. Additionally, it supports the nearest neighbor query well, which will be required to perform the threshold queries efficiently. Let us note that each object is represented by several points in the time-interval plane. Consequently, each object is referenced by the index structure multiple times. This property has to be taken into account when designing an efficient query algorithm. Section 4.5 covers the query algorithm in greater detail.

4.4.2 Representing Threshold-Crossing Time Intervals for Arbitrary Threshold Values

In contrast to the first approach presented above we will now describe how to manage threshold queries for arbitrary threshold values τ efficiently. First, we extend the transformation task of the basic approach in such a way that the time-interval plane representation of the threshold-crossing time interval sequences are available for all possible threshold values τ. Therefore, we extend the time-interval plane by one additional dimension corresponding to the threshold value. In the following, we denote this space *parameter space.* A two-dimensional plane parallel to the (lower,upper)-plane at a certain threshold τ in the parameter space is called time-interval plane of threshold τ.

Lemma 4.1.
Let $X \in \mathcal{D}$ be a time series and $S_{\tau_1,X}$ and $S_{\tau_2,X}$ be two threshold-crossing time interval sequences of X, where w.l.o.g. $\tau_1 < \tau_2$. Let $s_1 \in S_{\tau_1,X}$ and $s_2 \in S_{\tau_2,X}$

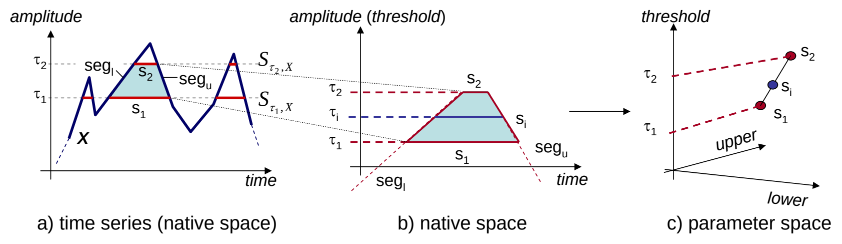


FIGURE 4.5: Parameter space for arbitrary threshold values.

*be two time intervals whose start points lie on a segment seg_l of the linearly interpolated time series and the end points lie on another segment seg_u. Then all threshold-crossing time interval sequences $S_{\tau_i,X}$ with $\tau_1 \leq \tau_i \leq \tau_2$ contain exactly one time interval $s_i \in S_{\tau_i,X}$ which also starts at segment seg_l and ends on segment seg_u. Transformed into the parameter space, s_i lies on the three-dimensional straight line: $g_P : \overrightarrow{x} = \overrightarrow{p_1} + \triangle t \cdot (\overrightarrow{p_2} - \overrightarrow{p_1})$,
where $\overrightarrow{p_1} = (s_1.lower, s_1.upper, \tau_1)^T$ and $\overrightarrow{p_2} = (s_2.lower, s_2.upper, \tau_2)^T$.*

Proof. *Both, the start point and the end point of s_i linearly depend on the threshold τ_i. Consequently, all s_i lie on a three-dimensional straight line in the parameter space. Let $\triangle t = (\tau_i - \tau_1)/(\tau_2 - \tau_1)$. Then,*

$$s_i = (s_i.lower, s_i.upper, \tau_i),$$

where

$$s_i.lower = s_1.lower + \triangle t \cdot (s_2.lower - s_1.lower),$$

$$s_i.upper = s_1.upper + \triangle t \cdot (s_2.upper - s_1.upper)$$

and

$$\tau_i = \tau_1 + \triangle t \cdot (\tau_2 - \tau_1).$$

$\square$

Let us consider the following example shown in Figure 4.5 in order to clarify Lemma 4.1. Figure 4.5(a) shows a linearly interpolated time series X. Let $s_1 \in S_{\tau_1,X}$ and $s_2 \in S_{\tau_2,X}$ be two time intervals. s_1 and s_2 are left bounded by the time series segment seg_l and right bounded by seg_u. All threshold-crossing time interval sequences $S_{\tau_i,X}$ which are between $S_{\tau_1,X}$ and

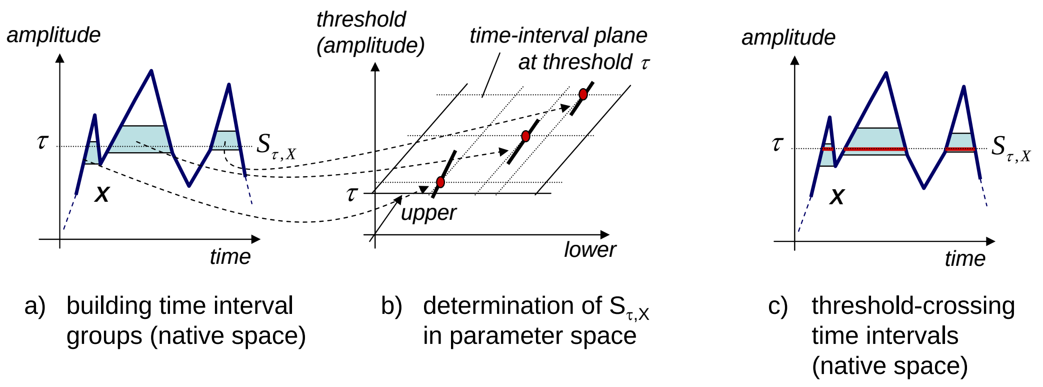

amplitude
τ
X
$S_{\tau,X}$
time
a) building time interval groups (native space)
threshold (amplitude)
time-interval plane at threshold τ
τ
upper
lower
b) determination of $S_{\tau,X}$ in parameter space
amplitude
τ
X
$S_{\tau,X}$
time
c) threshold-crossing time intervals (native space)

FIGURE 4.6: Using the parameter space to derive threshold-crossing time intervals for a given threshold.

$S_{\tau_2,X}$, i.e. $\tau_1 \leq \tau_i \leq \tau_2$ contain exactly one time interval s_i which is also bounded by the time series segments seg_l and seg_u as depicted in Figure 4.5(b). If s_1 and s_2 are mapped to the parameter space, the time interval s_i lies on the straight line between s_1 and s_2 in the parameter space as depicted in Figure 4.5(c).

Following Lemma 4.1, all time intervals which are bounded by the same time series segments can be transformed into the same segment in the parameter space. In order to represent all threshold-crossing time intervals of a time series in the parameter space, we have to identify all groups of time intervals where each group contains those time intervals which are bounded by the same time series segment in the native space (cf. Figure 4.6(a)). Each group then corresponds to a three-dimensional segment in the parameter space (cf. Figure 4.6(b)).

The complete set of all possible threshold-crossing time intervals of a time series X is represented as a set of segments in the parameter space. The time intervals which correspond to a certain threshold-crossing time interval sequence $S_{\tau,X}$ can be calculated by intersecting the parameter-space segments corresponding to X with the two-dimensional time-interval plane for threshold τ (cf. Figure 4.6(b)). The resulting intersection points correspond to the time intervals of $S_{\tau,X}$ as depicted in Figure 4.6(c).

We can efficiently manage the complete set of threshold-crossing time intervals in the parameter space as follows:

- We represent the set of threshold-crossing time intervals by the smallest possible number of segments in the parameter space.
- We organize the resulting parameter-space segments by means of a spatial index structure, e.g. the R*-tree.

In the following, we introduce a method which enables us to efficiently compute the smallest number of parameter-space segments for a given time series.

4.4.3 Trapezoid Decomposition of Time Series

Considering the possible threshold-crossing time intervals, we can use the following property.

Lemma 4.2.
Threshold-crossing time intervals always start at increasing time series segments (positive segment slope) and end at decreasing time series segments (negative segment slope).

Proof. *According to Definition 4.1, all values of X within the threshold-crossing time interval sequence $S_{\tau,X}$ are larger than the corresponding threshold value τ. Let us assume that the time series segment seq_l which lower-bounds the time interval at time t_l has a negative slope. Then all $x(t)$ on s_l with $t > t_l$ are smaller than τ which contradicts the definition of threshold-crossing time intervals. The validity of Lemma 4.2 w.r.t. the right bounding segment can be shown analogously.* $\square$

According to Lemma 4.2, the set of all time intervals which start and end at the same time series segment seg_l and seg_u respectively, can be described by a single trapezoid whose left and right bounds are congruent with seg_l and seg_u. Let $seg_l = ((x_{l1}, t_{l1}), (x_{l2}, t_{l2}))$ denote the segment of the left bound

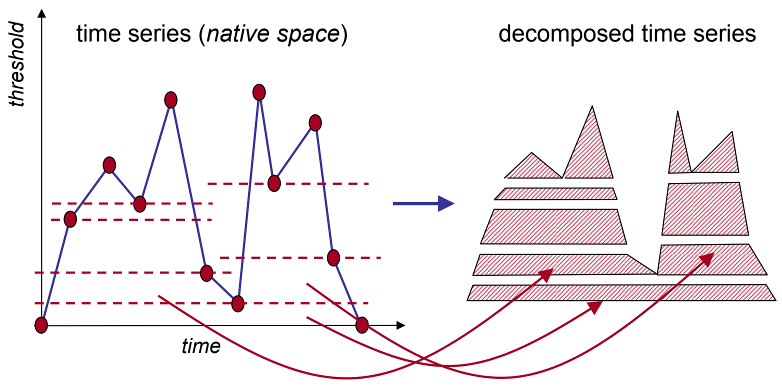


Figure 4.7: Time Series Decomposition

and $seg_u = ((x_{u1}, t_{u1}), (x_{u2}, t_{u2}))$ denote the segment of the right bound. The top-bottom bounds correspond to the two time intervals $s_{\tau_{top}}$ and $s_{\tau_{bottom}}$ at the threshold values:

$$\tau_{top} = \min(\max(x_{l1}, x_{l2}), \max(x_{r1}, x_{r2}));$$

$$\tau_{bottom} = \max(\min(x_{l1}, x_{l2}), \min(x_{r1}, x_{r2}));$$

In order to determine the minimal but complete set of parameter space segments of a time series, we have to determine the minimal set of trapezoids completely covering all possible threshold-crossing time intervals. The optimal set of trapezoids can be determined by decomposing the area below the time series into a set of disjoint trapezoids. A time series object can be considered as half-open uni-monotone polygon in the time-amplitude plane. There exist several sweep-line based polygon-to-trapezoid decomposition algorithms [FM84] of time complexity $O(n \cdot logn)$ in the number of vertices. We adopted one of these decomposition algorithms. Figure 4.7 shows an example of how a time series is decomposed into the set of trapezoids.

As we can assume that the time series consist of chronologically ordered pairs (x, t), our decomposition algorithm can be performed in linear time with respect to the length of the time series. The decomposition algorithm is given in Figure 4.9 and in Figure 4.10.

Let us illustrate the decomposition algorithm by means of the example

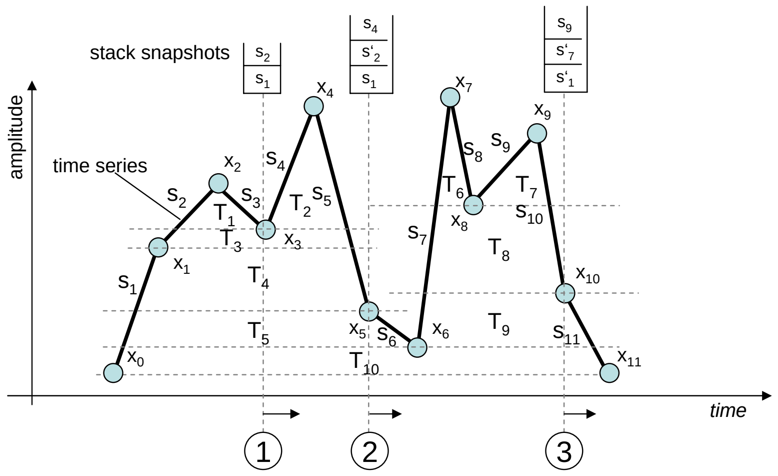


FIGURE 4.8: Time Series Decomposition Example.

depicted in Figure 4.8. In the *for*-loop we sequentially process the time series segments s_1,..,s_{11}. As s_1 and s_2 have positive slopes we put them on top of the stack. Next, we consider the segment $next\_seg = s_3$ which has a negative slope, i.e. we can close the first trapezoids. Actually (see step (1)), the stack contains the segments s_2, s_1. We pop s_2 from the stack and compute the first trapezoid T_1 by means of the procedure *compute_trapezoid*(s_2,s_3). Then we intersect the segment s_2 at the amplitude value $s_3.x_e = x_3$ and push the split segment s_2 denoted by s_2' back on the stack. We continue with the next segment s_4 which is pushed on the stack. Next, we proceed segment s_5 by taking s_4 from stack, compute the trapezoid T_2, then taking s_2' from stack in order to compute T_3 and finally taking s_1 from stack, compute T_4, split s_1 w.r.t. x_5 and push the split segment s_5' back on the stack.

4.4.4 Indexing Segments in the Parameter Space

We apply the R*-tree for the efficient management of the three-dimensional segments representing the time series objects in the parameter space. As the R*-tree index can only manage rectangles, we represent the 3-dimensional segments by rectangles where the segments correspond to one of the diagonals of the rectangles.

```
TYPE TSSegment = {start time $t_s$, start value $x_s$, end time $t_e$, end value $x_e$};
decompose(time series $TS = \{(x_i, t_i) : i = 0..t_{max}\}$){
  /*initialize start and end point of the time series*/
  stack.push(TSSegment($t_0, \perp, t_0, x_0$)); //left time series border on stack
  TS.append(($t_{max}$,$\perp$)); //append right time series border
  for $i = 1..t_{max}$ do
     next_seg := TSSegment($t_{i-1}, x_{i-1}, t_i, x_i$);
     if ($x_{i+1} < x_i$), then //segment with positive slope $\Rightarrow$ open trapezoid
       stack.push(next_seg);
     else if ($x_{i+1} > x_i$), then //segment with negative slope $\Rightarrow$ close trapezoids
       while (stack.top.$x_s \geq$ next_seg.$x_e$) do
         stack_seg = stack.pop();
         compute_trapezoid(stack_seg,next_seg);
         end while;
       stack_seg = stack.pop();
       compute_trapezoid(stack_seg,next_seg);
       stack_seg = cut_segment_at(next_seg.$x_e$);
       stack.push(stack_seg);
     else /*nothing to do*/; //horizontal segment => can be ignored
     end if;
  end for;
}
```

FIGURE 4.9: Linear time series decomposition.

```
TYPE Trapezoid = {bottom start (Time), bottom end (Time), bottom (float), top start
(Time), top end (Time), top (float)};
compute_trapezoid(TSSegment seg1, TSSegment seg2){
  float $\tau_{bottom}$ = max(seg1.$x_s$,seg2.$x_e$);
  float $\tau_{top}$ = min(seg1.$x_e$,seg2.$x_s$);
  Time $t_s^{bottom}$ = intersect(seg1,$\tau_{bottom}$);
  Time $t_e^{bottom}$ = intersect(seg2,$\tau_{bottom}$);
  Time $t_s^{top}$ = intersect(seg1,$\tau_{top}$);
  Time $t_e^{top}$ = intersect(seg2,$\tau_{top}$);
  output(Trapezoid($t_s^{bottom}$,$t_e^{bottom}$,$\tau_{bottom}$,$t_s^{top}$,$t_e^{top}$,$\tau_{top}$));
}
```

FIGURE 4.10: Auxiliary function for the linear time series decomposition.

For all trapezoids which result from the time series decomposition, the lower bound time interval contains the upper bound time interval. Furthermore, intervals which are contained in another interval are located in the lower-right area of this interval representation in the time-interval plane. Consequently, the locations of the segments within the rectangles in the parameter space are fixed. Therefore, in the parameter space the bounds of the rectangle which represents a segment are sufficient to uniquely identify the covered segment. Let $((x_l, y_l, z_l), (x_u, y_u, z_u))$ be the coordinates of a rectangle in the parameter space. Then the coordinates of the corresponding segment are $((x_l, y_u, z_l), (x_u, y_l, z_u))$.

4.5 Threshold-Based Query Algorithms

In this section, we present efficient algorithms for the two threshold queries, the threshold-based ε-range query and the threshold-based k-nearest-neighbor query.

A straightforward approach for the query algorithm is the following: first, we access all parameter space segments of the database objects which intersect the time-interval plane at threshold τ by means of the R^*-tree index in order to retrieve the threshold-crossing time intervals of all database objects. Then, for each database object we compute the τ-similarity to the query object. We only have to access the relevant parameter space segments instead of accessing the entire object. But we can process threshold queries in a more efficient way. In particular, for selective queries we do not need to access all parameter space segments of all time series objects covering the threshold amplitude τ. We can achieve a better query performance by using the R*-tree index to prune the segments of those objects which cannot satisfy the query anymore as early as possible.

4.5.1 Preliminaries

In the following, we assume that each time series object $X \in \mathcal{D}$ is represented by its threshold-crossing time intervals $S_X = S_{\tau,X} = x_1, .., x_N$ which correspond to a set of points in the time-interval plane $\mathcal{P}$. $\mathcal{P}$ is a plane of the parameter space at query threshold τ. Hence, S_X denotes a set of two-dimensional points. Furthermore let $\mathcal{S}$ denote the set of all time-interval points on $\mathcal{P}$ derived from all threshold-crossing time intervals $S_{\tau,X}$ of all objects $X \in \mathcal{D}$.

For our approach, we require two basic set operations on single time interval data (represented as points on the time-interval plane $\mathcal{P}$), the *ε-range set* and the *k-nearest-neighbor* which are defined as follows:

Definition 4.6 (ε-Range Set).
Let $q \in \mathcal{P}$ be a time interval, $S = \{x_i : i = 1..N\} \subseteq \mathcal{P}$ be a set of N time intervals and $\varepsilon \in \mathbb{R}_0^+$ be the maximal similarity-distance parameter. Then the ε-range set *of q is defined as follows:*

$$R_{\varepsilon,S}(q) = \{s \in S | d_{int}(s, q) \leq \varepsilon\}.$$

Definition 4.7 (k-Nearest-Neighbor).
Let $q \in \mathcal{P}$ be a time interval, $S = \{s_i : i = 1..N\} \subseteq \mathcal{P}$ be a set of N time intervals and $k \in \mathbb{N}^+$ be the ranking parameter. The k-nearest-neighbor *element $NN_{k,S}(q) \in \mathcal{P}$ $(k \leq N)$ of q in the set S is defined as follows:*

$$s = NN_{k,S}(q) \in S \Leftrightarrow \forall s' \in S \backslash \{NN_{l,S}(q) : l \leq k\} : d_{int}(q, s) \leq d_{int}(q, s').$$

The distance $d_{int}(q, NN_{k,S}(q))$ is called k-nearest-neighbor distance. *For $k = 1$, we simply call $NN_{1,S}(q) \equiv NN_S(q) \in \mathcal{P}$ the nearest-neighbor of q in S. The set $kNN_S(q) = \{NN_{l,S_X}(q) | l = 1..k\} \subseteq \mathcal{P}$ is called* k-nearest-neighbors *of q.*

Table 4.1 summarizes the most important parameters required throughout the following sections.

$\mathcal{P}$	Time-interval plane for query threshold τ.
$\mathcal{S}$	Set of all time intervals $\in S_{\tau,X} \subseteq \mathcal{P}$ of all time series objects in $\mathcal{D}$.
$R_{\varepsilon,S}(q)$	Set of time intervals from S which belongs to the ε-*range set* of q (cf. Definition 4.6).
$NN_S(q)$	The nearest neighbor of q in S (cf. Definition 4.7).
$NN_{k,S}(q)$	The k^{th} nearest neighbor of q in S (cf. Definition 4.7).
$kNN_S(q)$	The k nearest neighbors of q in S (cf. Definition 4.7).

Table 4.1: Important notations for time interval sets.

4.5.2 Pruning Strategy for Threshold Queries

In order to compute threshold queries, we do not have to access all time intervals in $\mathcal{S}$. Rather, we can prune objects without accessing them. The pruning strategy is based on the following observations.

Lemma 4.3 (Pruning Condition for Range Queries).
Let $S_Q \subseteq \mathcal{P}$ be the points corresponding to the query object Q. Then, each database object $X \in \mathcal{D}$ represented by $S_X \subseteq \mathcal{P}$ which has no time interval $s \in S_X$ in the ε-range of one of the query time intervals $q \in S_Q$ cannot belong to the result of the threshold-based ε-range query $TQ_\varepsilon^{range}(Q, \tau)$, i.e.

$$\forall s \in S_X, \forall q \in S_Q : s \notin R_\varepsilon(q) \Rightarrow X \notin TQ_\varepsilon^{range}(Q, \tau).$$

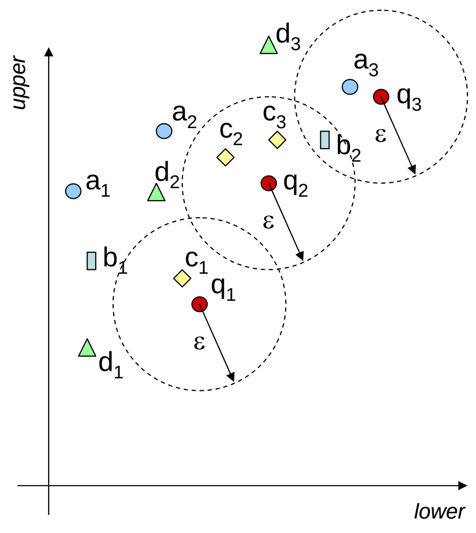
Proof. *Let $X \in \mathcal{D}$ be the database object which has no time interval $s \in S_X$ in the ε-range of one of the query time intervals $q \in S_Q$. That means that*

$$\forall s \in S_X, \forall q \in S_Q : d_{int}(s, q) > \varepsilon.$$

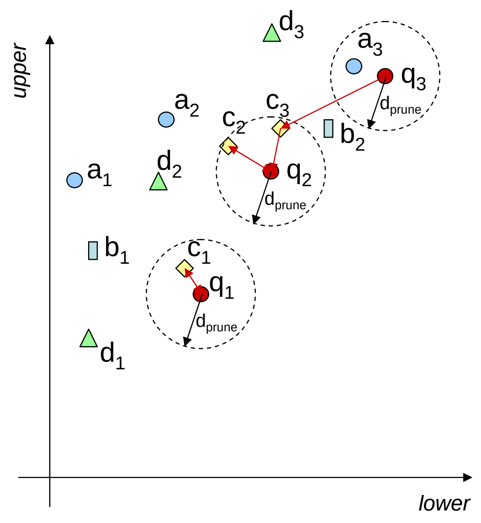
Then the following statement holds:

$$d_{TS}(S_Q, S_X) = \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot \sum_{q \in S_Q} \min_{s \in S_X} d_{int}(q, s) + \frac{1}{|S_X|} \cdot \sum_{s \in S_X} \min_{q \in S_Q} d_{int}(s, q) \right)$$

$$> \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot \sum_{q \in S_Q} \varepsilon + \frac{1}{|S_X|} \cdot \sum_{s \in S_X} \varepsilon \right) = \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot |S_Q| \cdot \varepsilon + \frac{1}{|S_X|} \cdot |S_X| \cdot \varepsilon \right)$$



(a) threshold-based ε-range query



(b) threshold-based k-nearest-neighbor query

FIGURE 4.11: Pruning techniques for threshold queries.

The last term equals ε and so the lemma is valid. □

An example is depicted in Figure 4.11(a) showing the threshold-crossing time intervals $S_Q = \{q_1, q_2, q_3\}$ for the query object Q and the threshold-crossing time intervals of the four database objects A, B, C, and D. Following Lemma 4.3, object D cannot be in the result set of $TQ_{\varepsilon}^{range}(Q, \tau)$.

Analogously we can identify pruning candidates for the k-nearest-neighbor query. For the sake of the presentation, we assume the ranking parameter k to be 1.

Lemma 4.4 (Pruning Condition for kNN Queries).
Let $S_Q \subseteq \mathcal{P}$ be the points corresponding to the query object Q. Furthermore, let d_{prune} be the threshold distance $d_{TS}(S_Q, S_X)$ between Q and any database object X. Then each database object $Y \in \mathcal{D}$ represented by $S_Y \subseteq \mathcal{P}$ which has no time interval $s \in S_Y$ in the d_{prune}-range of one of the query time intervals $q \in S_Q$, cannot belong to the result of the threshold-based k-nearest-neighbor

query $TQ_1^{NN}(Q,\tau)$, formally:

$$\forall s \in S_Y, \forall q \in S_Q : s \notin R_{d_{prune}}(q) \Rightarrow Y \notin TQ_1^{NN}(Q,\tau).$$

Proof. *Let $Y \in \mathcal{D}$ be the database object which has no time interval $s \in S_Y$ in the d_{prune}-range of one of the query time intervals $q \in S_Q$. That means that*

$$\forall s \in S_Y, \forall q \in S_Q : d_{int}(s,q) > d_{prune}.$$

Then the following statement holds:

$$d_{TS}(S_Q,S_Y) = \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot \sum_{q \in S_Q} \min_{s \in S_Y} d_{int}(q,s) + \frac{1}{|S_Y|} \cdot \sum_{s \in S_Y} \min_{q \in S_Q} d_{int}(s,q) \right)$$

$$> \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot \sum_{q \in S_Q} d_{prune} + \frac{1}{|S_Y|} \cdot \sum_{s \in S_X} d_{prune} \right)$$

$$= \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot |S_Q| \cdot d_{prune} + \frac{1}{|S_Y|} \cdot |S_Y| \cdot d_{prune} \right) = d_{prune} = d_{TS}(S_Q,S_X).$$

According to Definition 4.5, Y cannot be in the result set $TQ_1^{NN}(Q,\tau)$. □

An example for Lemma 4.4 is given in Figure 4.11(b). Object B cannot be a result of $TQ_1^{NN}(Q,\tau)$, because all distances $d_{int}(q,b)$ between any time interval q of S_Q and any time interval b of S_B exceed d_{prune}.

Based on the two lemmas above, we can develop efficient threshold queries using the R^*-tree .

4.5.3 Threshold-Based ε-Range Query Algorithm

The algorithm for the threshold-based ε-range query is depicted in Figure 4.12. We assume that the threshold-crossing time intervals of the query object Q are already available. The algorithm follows the filter-refinement paradigm: in a filter step, we retrieve the ε-range set $R_{\varepsilon,\mathcal{S}}(q)$ for each time interval $q \in S_Q$ by means of the R^*-tree and determine the corresponding time series candidate set. Afterwards, in the refinement step we refine each candidate X by computing the exact threshold distance to Q.

```
ALGORITHM $TQ^{range}(S_Q, \varepsilon, \mathcal{D}, \mathcal{S})$
  result_set := $\emptyset$;
  candidate_set := $\emptyset$;
  FOR EACH $q \in S_Q$ DO
    candidate_set := candidate_set $\cup \{X \in \mathcal{D} | S_X \cap R_{\varepsilon,\mathcal{S}}(q) \neq \emptyset\}$; // filter step
  END FOR;
  FOR EACH $X \in$ candidate_set DO
    IF $d_{TS}(S_Q, S_X) \leq \varepsilon$ THEN // refinement step
      result_set := result_set $\cup X$;
  END FOR;
  export result_set;
```

FIGURE 4.12: Threshold-based ε-range query algorithm.

4.5.4 Filter Distance for the Threshold Similarity

For the k-nearest-neighbor query algorithm, a suitable filter distance for the pruning strategy is required.

Lower Bounding Threshold Distance

In the following we introduce a lower bound criterion for the threshold distance d_{TS} based on partial distance computations between the query object and the database objects. This lower bound criterion enables the detection of false candidates (true drops) very early. The amount of information which is necessary to prune an object depends on the locations of the query object and the candidate objects.

In the following, we assume that $S_Q \subseteq \mathcal{P}$ is the set of threshold-crossing time intervals for the query object and $S_X \subseteq \mathcal{P}$ is the set of threshold-crossing time intervals for any object X from the database. Furthermore, we need the following two distance functions

$$D_1(S_Q, S_X) = \sum_{q \in S_Q} d_{int}(q, NN_{S_X}(q))$$

and

$$D_2(S_Q, S_X) = \sum_{x \in S_X} d_{int}(x, NN_{S_Q}(x)).$$

$D_1(S_Q, S_X)$ and $D_2(S_Q, S_X)$ are parts of the threshold distance which can be expressed as:

$$d_{TS}(S_Q, S_X) = \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot D_1(S_Q, S_X) + \frac{1}{|S_X|} \cdot D_2(S_Q, S_X) \right).$$

We use two auxiliary functions $\kappa_k(q_i)$ and $\bar{\kappa}_k(S_Q)$. We use this functions to partition the database objects into two sets. $\kappa_k(q_i) \subseteq \mathcal{D}$ denotes the set of all objects X which have at least one element $x \in S_X$ within the set $kNN_X(q_i)$. Furthermore, $\bar{\kappa_k}(S_Q) \subseteq \mathcal{D}$ denotes the set of all objects which are not in any set $\kappa_k(q_i)$, i.e. $\bar{\kappa}_k(S_Q) = \mathcal{D} \backslash (\bigcup_{q \in S_Q} \kappa_k(q))$.

Lemma 4.5.
For any object $X \in \bar{\kappa}_k(S_Q)$ the following inequality holds :

$$D_1(S_Q, S_X) \geq \sum_{q \in S_Q} d_{int}(q, NN_{k,\mathcal{S}}(q)).$$

Proof. *According to Definition 4.7 the following statement holds:*

$$\forall q \in S_Q : d_{int}(q, NN_{k,\mathcal{S}}(q)) \leq d_{int}(q, NN_{S_X}(q)).$$

Therefore,

$$\sum_{q \in S_Q} d_{int}(q, NN_{k,\mathcal{S}}(q)) \leq \sum_{q \in S_Q} d_{int}(q, NN_X(q)) = D_1(S_Q, S_X).$$

□

The next lemma is a generalization of Lemma 4.5 and defines a lower bound of $D_1(S_Q, S_X)$ for all database objects $X \in \mathcal{D}$ for any $k \in \mathbb{N}^+$.

Lemma 4.6 (Lower Bound for D_1).
Let $X \in \mathcal{D}$ be any database object and let Q be the query object. The distance $D_1(S_Q, S_X)$ can be lower-bounded by:

$$d_1^{min}(S_Q, S_X) = \sum_{q \in S_Q} \left\{ \begin{array}{ll} d_{int}(q, NN_X(q)), & \text{if } X \in \kappa_k(q) \\ d_{int}(q, NN_{k,\mathcal{S}}(q)), & \text{else} \end{array} \right\} \leq D_1(S_Q, S_X).$$

Proof. *Let $X \in \mathcal{D}$ be any database object and Q be the query object. According to Definition 4.7 the following holds:*

$$\forall q \in S_Q : X \notin \kappa_k(q) \Rightarrow d_{int}(q, NN_{k,\mathcal{S}}(q)) \leq d_{int}(q, NN_X(q)).$$

Consequently, $d_1^{min}(Q, X) \leq \sum_{q \in S_Q} d_{int}(q, NN_X(q)) = D_1(S_Q, S_X)$. □

$D_2(S_Q, S_X)$ can be lower-bounded using the next lemma.

Lemma 4.7 (Lower Bound for D_2).
Let $X \in \mathcal{D}$ be any database object and let Q be the query object. The distance $D_2(S_Q, S_X)$ can be estimated by the following formula:

$$d_2^{min}(S_Q, S_X) =$$

$$\min_{q \in S_Q} \left\{ \begin{array}{ll} d_{int}(q, NN_X(q)), & \text{if } d_{int}(q, NN_X(q)) < d_{int}(q, NN_{k,\mathcal{S}}(q)) \\ d_{int}(q, NN_{k,\mathcal{S}}(q)), & \text{else} \end{array} \right\}$$

$$\leq \frac{1}{|S_X|} \cdot D_2(S_Q, S_X).$$

Proof. *Let $X \in \mathcal{D}$ be any database object and Q be the query object. Generally, the following statement holds:*

$$\min_{q \in S_Q}(d_{int}(q, NN_{S_X}(q))) = \min_{s \in S_X}(d_{int}(s, NN_{S_Q}(s))) \leq \frac{1}{|S_X|} \cdot D_2(S_Q, S_X).$$

If $\forall q \in S_Q : NN_X(q) \geq \min_{q \in S_Q}(NN_{k,\mathcal{S}}(q))$, then all time intervals $s \in S_X$ must have at least the distance to any $q \in S_Q$ which is greater or equal to the smallest k-nearest-neighbor distance of any $q \in S_Q$, i.e.

$$\forall s \in S_X, \forall q \in S_Q : d_{int}(q, s) \geq NN_{k,\mathcal{S}}(q) \geq \min_{q \in S_Q} d_{int}(q, NN_{k,\mathcal{S}}(q)).$$

With the equation above and Definition 4.7 the following statement holds:

$$\forall s \in S_X : d_{int}(s, NN_{S_Q}(s)) \geq \min_{q \in S_Q} d_{int}(q, NN_{k,\mathcal{S}}(q))$$

which obviously holds also for the average nearest-neighbor distance of all $s \in S_X$*, i.e.*

$$\frac{1}{S_X} \cdot \sum_{s \in S_X} d_{int}(s, NN_{S_Q}(s)) = \frac{1}{S_X} \cdot D_2(S_Q, S_X) \geq \min_{q \in S_Q} d_{int}(q, NN_{k,\mathcal{S}}(q)).$$

□

Lower-Bound-Based Pruning

In this section, we show which objects can be pruned, based on the information retrieved so far. Let us assume that $Q \in \mathcal{D}$ is the query object, $X \in \mathcal{D}$ is any object which has been already refined, i.e. $d_{TS}(Q, X)$ is known and $Y \in \mathcal{D}$ is another object which has not yet been refined. Then we can prune Y for the threshold query $TQ_1^{k-NN}(Q, \tau)$ if and only if:

$$\begin{aligned} & d_{TS}(S_Q, S_Y) > d_{TS}(S_Q, S_X) \\ \Leftrightarrow \; & \frac{1}{|S_Q|} D_1(S_Q, S_Y) + \frac{1}{|S_Y|} D_2(S_Q, S_Y) > 2 \cdot d_{TS}(S_Q, S_X) \\ \Leftrightarrow \; & D_1(S_Q, S_Y) + \frac{|S_Q|}{|S_Y|} \cdot D_2(S_Q, S_Y) > 2 \cdot |S_Q| \cdot d_{TS}(S_Q, S_X). \end{aligned}$$

Applying Lemma 4.6 and 4.7, Y can be pruned if and only if

$$d_1^{min}(S_Q, S_Y) + |S_Q| \cdot d_2^{min}(S_Q, S_Y) > 2 \cdot |S_Q| \cdot d_{TS}(S_Q, S_X).$$

In the following, let $d_{prune} = 2 \cdot |S_Q| \cdot d_{TS}(S_Q, S_X)$ be the pruning distance.

From a computational point of view, we should distinguish the objects in $\bar{\kappa}_k(S_Q)$ from the other objects. The next two statements follow directly from Lemma 4.6 and Lemma 4.7:

Lemma 4.8.
All objects $Y \in \bar{\kappa}_k(S_Q)$*, can be pruned if and only if*

$$\sum_{q \in S_Q} d_{int}(q, NN_{k,\mathcal{S}}(q)) + |S_Q| \cdot \min_{q \in S_Q} d_{int}(q, NN_{k,\mathcal{S}}(q)) > d_{prune}.$$

Lemma 4.9.
All objects $Y \notin \bar{\kappa}_k(S_Q)$, can be pruned if and only if

$$d_1^{min}(S_Q, S_Y) + |S_Q| \cdot \min_{q \in S_Q}(\min(d_{int}(q, NN_{k,\mathcal{S}}(q)), d_{int}(q, NN_Y(q))) > d_{prune}.$$

Our query procedure is based on an iterative ranking query for each query time interval $q \in S_Q \subseteq \mathcal{P}$, i.e. we iteratively compute the k-nearest-neighbors $NN_{k,\mathcal{S}}(q) \subseteq \mathcal{P}$ for all $q \in S_Q$ with increasing $k \in \mathbb{N}^+$. After each iteration, we determine the lower bound distances for all objects. Note that we only need to materialize the partial distance information for those objects which are not in $\bar{\kappa}_k(S_Q)$, i.e. for those objects for which we have retrieved at least one time interval so far. These objects are organized in a list which might be expanded in each iteration. This list is called *object list*. Now, we can compute the lower bounding distance for all objects in the object list and prune them according to Lemma 4.9. The lower bounding distance estimation for all other objects can be computed with global parameters, in particular $d_{int}(q, NN_{k,\mathcal{S}}(q)$ (cf. Lemma 4.8). As soon as we have found a pruning distance d_{prune} for which Lemma 4.8 holds, we do not need to expand the object list anymore.

At the moment, we have found the nearest neighbor of each $q \in S_Q$ w.r.t. any database object X, i.e. $\forall q \in S_Q : S_X \in \kappa_k(q)$, the lower bound distance $d_1^{min}(S_Q, S_X)$ is equal to $D_1(S_Q, S_X)$. Then, both lower bound distances d_1^{min} and d_2^{min} cannot be improved by further query iterations. For this reason, we refine the distance $d_{TS}(S_Q, S_X)$ by accessing the complete threshold-crossing time intervals S_X in order to exactly compute the distance $D_2(S_Q, S_X)$. The resulting distance $d_{TS}(S_Q, S_X)$ is then used as new pruning distance d_{prune} for the remaining query process unless $d_{TS}(S_Q, S_X)$ is lower than the old pruning distance. Let X be the object with the lowest exact distance to Q, i.e. $d_{prune} = 2 \cdot |S_Q| \cdot d_{TS}(S_Q, S_X)$. The pruning distance may be updated as soon as an object S_Y which has to be refined next is found. We have to consider two cases:

case 1: $2 \cdot |S_Q| \cdot d_{TS}(S_Q, S_Y) \geq d_{prune} \rightarrow$ remove object S_Y from the candidate set

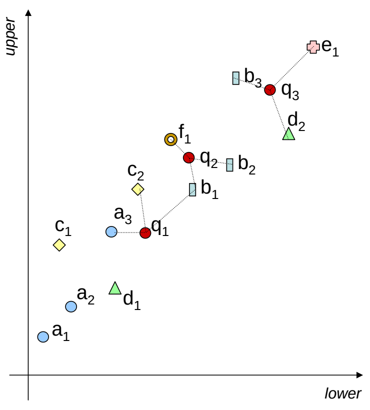
case 2: $2 \cdot |S_Q| \cdot d_{TS}(S_Q, S_Y) < d_{prune} \rightarrow$ set $d_{prune} := 2 \cdot |S_Q| \cdot d_{TS}(S_Q, S_Y)$ and remove object S_X from the candidate set.

After each query iteration, we prune all objects $Y \in \mathcal{D} \backslash \{X\}$ from the object list according to Lemma 4.9. The search proceeds by continuing the computation of the next ranking iteration $NN_{k+1,\mathcal{S}}$. The search algorithm terminates as soon as all object candidates, except for the most similar one (in case of the threshold-based 1^{st}-nearest-neighbor query), have been pruned from the object list.

4.5.5 Threshold-Based Nearest-Neighbor Query Algorithm

The query algorithm of the TQ_1^{NN} query is depicted in Figure 4.15. It iteratively computes for a given query object S_Q the database object X, having the smallest threshold distance $d_{TS}(S_Q, S_X)$. In each iteration (repeat-loop), we retrieve the next ranked time interval $s \in \mathcal{S}$ (k-nearest-neighbor) for each $q \in S_Q$ by calling the function *fetch-next()* and store it with its distance to q in the array *act_kNN*. This can be efficiently done by applying the nearest neighbor ranking method as proposed in [HS95]. For each $q \in S_Q$ we maintain a priority queue, storing the visited R^*-tree nodes in ascending order with respect to their distances to the corresponding query point q. Note that the R^*-tree indexes the three-dimensional segments in the parameter space, although we are only interested in distances along the time-interval plane at threshold τ. For this reason, we simply ignore the threshold-dimension for the distance computations and consider only those R^*-tree nodes intersecting the time-interval plane at threshold τ. Obviously, the ranking function only considers those objects which were not already pruned from the object list and which cannot be pruned according to Lemma 4.8.

Furthermore, we update the object list *object_distList* maintaining for each already accessed X an array. This array stores for each $q \in S_Q$ the nearest-neighbor distance $NN_X(q)$ in case this information is already available. For this reason, for each time interval s retrieved by $NN_{k,\mathcal{S}}(q)$, we



(a) Time interval instances on the time-interval plane $\mathcal{P}$

iterative computation of $NN_{k,S}(q_i)$

	q_1	q_2	q_3
$NN_{1,S}(q_i)$	a_3	f_1	b_3
$NN_{2,S}(q_i)$	c_2	b_1	d_2
$NN_{3,S}(q_i)$	b_1	b_2	e_1

(b) Table of nearest-neighbor query iterations

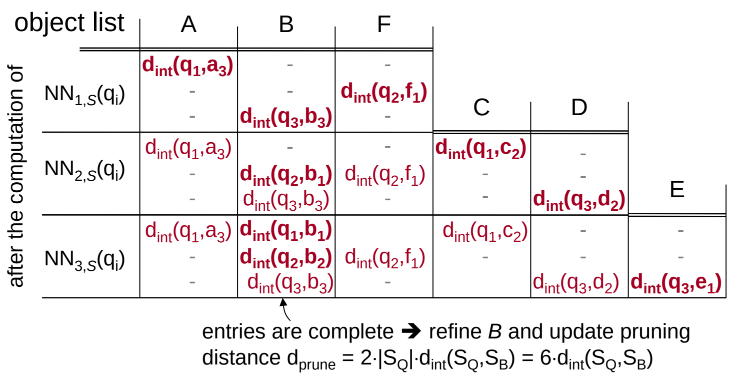
FIGURE 4.13: Example for the threshold-based nearest-neighbor query.

determine the corresponding object in the object list *object_distList* and store the distance $d_{int}(s,q)$, if with respect to q and X there is no distance available from earlier iterations. As soon as we have retrieved all $NN_X(q)$-distances for all $q \in S_Q$ for an object X , we refine this object by accessing the full object information and computing the threshold distance $d_{TS}(S_Q, S_X)$. After the refinement, we can obviously update the pruning distance d_{prune}. If X is a nearer neighbor than the previous *result*, we replace the previous *result* with X. Next, we compute the lower-bounding distance *lb_dist* for each object in the object list and prune those objects for which *lb_dist*$\geq d_{prune}$ holds.

As long as the object list is not empty, we repeat this procedure in the next iterations. Finally, we have found the (1^{st}-)nearest-neighbor of Q, based on our threshold-based similarity measure.

In order to enable the computation of threshold-based k-nearest-neighbor queries, we have to modify our algorithm marginally. First, we have to keep the k closest objects w.r.t. the threshold distance during the query process. Instead of pruning the objects according to the distance of the currently closest object, we have to take the k closest object into account.

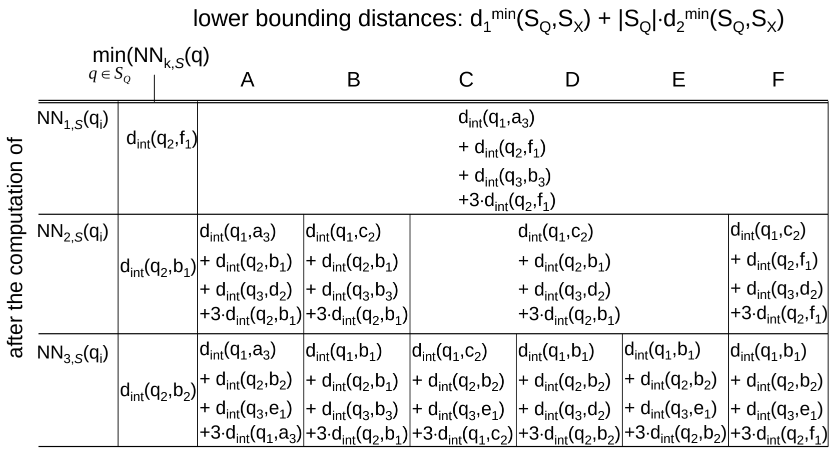
Figure 4.13 presents an example for our novel algorithm. The query con-



object list	A	B	F	C	D	E
$NN_{1,S}(q_i)$	$d_{int}(q_1,a_3)$ - -	- - $d_{int}(q_3,b_3)$	- $d_{int}(q_2,f_1)$ -			
$NN_{2,S}(q_i)$	$d_{int}(q_1,a_3)$ - -	- $d_{int}(q_2,b_1)$ $d_{int}(q_3,b_3)$	- $d_{int}(q_2,f_1)$ -	$d_{int}(q_1,c_2)$ - -	- - $d_{int}(q_3,d_2)$	
$NN_{3,S}(q_i)$	$d_{int}(q_1,a_3)$ - -	$d_{int}(q_1,b_1)$ $d_{int}(q_2,b_2)$ $d_{int}(q_3,b_3)$	- $d_{int}(q_2,f_1)$ -	$d_{int}(q_1,c_2)$ -	- - $d_{int}(q_3,d_2)$	- - $d_{int}(q_3,e_1)$

after the computation of
entries are complete ➔ refine B and update pruning distance $d_{prune} = 2 \cdot |S_Q| \cdot d_{int}(S_Q,S_B) = 6 \cdot d_{int}(S_Q,S_B)$

(a) Object list



lower bounding distances: $d_1^{min}(S_Q,S_X) + |S_Q| \cdot d_2^{min}(S_Q,S_X)$

	$\min_{q \in S_Q}(NN_{k,S}(q))$	A	B	C	D	E	F
$NN_{1,S}(q_i)$	$d_{int}(q_2,f_1)$	$d_{int}(q_1,a_3)$ $+ d_{int}(q_2,f_1)$ $+ d_{int}(q_3,b_3)$ $+3 \cdot d_{int}(q_2,f_1)$					
$NN_{2,S}(q_i)$	$d_{int}(q_2,b_1)$	$d_{int}(q_1,a_3)$ $+ d_{int}(q_2,b_1)$ $+ d_{int}(q_3,d_2)$ $+3 \cdot d_{int}(q_2,b_1)$	$d_{int}(q_1,c_2)$ $+ d_{int}(q_2,b_1)$ $+ d_{int}(q_3,b_3)$ $+3 \cdot d_{int}(q_2,b_1)$	$d_{int}(q_1,c_2)$ $+ d_{int}(q_2,b_1)$ $+ d_{int}(q_3,d_2)$ $+3 \cdot d_{int}(q_2,b_1)$			$d_{int}(q_1,c_2)$ $+ d_{int}(q_2,f_1)$ $+ d_{int}(q_3,d_2)$ $+3 \cdot d_{int}(q_2,f_1)$
$NN_{3,S}(q_i)$	$d_{int}(q_2,b_2)$	$d_{int}(q_1,a_3)$ $+ d_{int}(q_2,b_2)$ $+ d_{int}(q_3,e_1)$ $+3 \cdot d_{int}(q_1,a_3)$	$d_{int}(q_1,b_1)$ $+ d_{int}(q_2,b_1)$ $+ d_{int}(q_3,b_3)$ $+3 \cdot d_{int}(q_2,b_1)$	$d_{int}(q_1,c_2)$ $+ d_{int}(q_2,b_2)$ $+ d_{int}(q_3,e_1)$ $+3 \cdot d_{int}(q_1,c_2)$	$d_{int}(q_1,b_1)$ $+ d_{int}(q_2,b_2)$ $+ d_{int}(q_3,d_2)$ $+3 \cdot d_{int}(q_2,b_2)$	$d_{int}(q_1,b_1)$ $+ d_{int}(q_2,b_2)$ $+ d_{int}(q_3,e_1)$ $+3 \cdot d_{int}(q_2,b_2)$	$d_{int}(q_1,b_1)$ $+ d_{int}(q_2,b_2)$ $+ d_{int}(q_3,e_1)$ $+3 \cdot d_{int}(q_2,f_1)$

(b) Lower-bounding distance computation

Figure 4.14: Step-wise lower-bounding distance computation of the threshold-based nearest-neighbor query example.

sists of three time-interval plane points $S_Q = \{q_1, q_2, q_3\}$. Figure 4.13(a) shows the time-interval plane $\mathcal{P}$ with the three query time-interval points of S_Q and several time-interval points of six database objects. Figure 4.13(b) shows the results of the first three iterations of the incremental k-nearest-neighbor queries $NN_{1,\mathcal{S}}(q_i)$, $NN_{2,\mathcal{S}}(q_i)$ and $NN_{3,\mathcal{S}}(q_i)$. The state of the corresponding object list *object_distList* after each iteration is shown in Figure 4.14(a). Figure 4.14(b) depicts the lower bounding distances for each object after each query iteration.

The first iteration retrieves the points a_3, f_1, and b_3 of the objects A, F, and B, respectively. As a result, we add the objects A, B, and F to the object list and compute their lower bounding distances $d_1^{min}(S_Q, S_X) + |S_Q| \cdot d_2^{min}(S_Q, S_X)$ according to Lemma 4.9. In this case, all database objects have equal lower bounding distances as depicted in Figure 4.14(b). As the pruning distance d_{prune} is actually set to ∞, no object can be pruned.

In the next iteration, we retrieve c_1, b_1, and d_2, update the object list, and recompute the lower bounding distances. Next, we retrieve b_1, b_2, and e_1. After updating the object list, the entries are complete for object B, i.e. we have found for each query time-interval the corresponding nearest neighbor with respect to B. Consequently, we refine object B by accessing its complete set S_B and compute the exact threshold distance $d_{TS}(S_Q, S_B)$ in order to update the pruning distance d_{prune}. Afterwards, we remove object B from the object list and try to prune the other objects according to their lower bounding distances following Lemma 4.9 and 4.8.

The runtime complexity of our threshold query algorithm is $O(n_q \cdot n_k \cdot \log n_p)$, where n_q denotes the size of the threshold-crossing time interval sequence S_Q, n_k denotes the number of query iterations required to determine the query result, and n_p denotes the overall number of segments in the parameter space. In the experimental Section 4.6 we demonstrate that in average n_q is very small in comparison to the length of the time sequences. Furthermore, we show that the number of required nearest-neighbor query iterations n_k is small, i.e. the query process terminates early. The number n_p of segments in the parameter space is quite similar to the sum n_s of length of all time sequences in the database. We observed in our experiments that in fact n_p is slightly smaller than n_s.

4.6 Experimental Evaluation

In this section, we present the results of experiments performed on a broad selection of different time series datasets.

```
TYPE Q_ARRAY[N] := ARRAY[N] of DOUBLE;
ALGORITHM $TQ_1^{k-NN}(S_Q, \mathcal{D}, \mathcal{S})$
BEGIN
  act_kNN : ARRAY[$|S_Q|$] of (OID,DIST); /*current ranking status*/
  object_distList : LIST of (OID,DIST : Q_ARRAY[$|S_Q|$]); /*object list
  result := null;                                        with lb-distances*/
  $d_{prune} := +\infty$
  $k$ := 0;
  REPEAT
    $k := k + 1$;
    act_kNN = fetch-next($S_Q$,$\mathcal{S}$,$d_{prune}$);
    FOR $i = 1..|S_Q|$ DO
      $s$ := act_kNN[$i$].DIST;
      IF (s.oid not exists in object_distList) THEN
        object_distList.add(s.oid);
      END IF;
      IF (object_distList[s.oid].DIST[i] is empty) THEN
        object_distList[s.oid].DIST[i] := act_kNN[$i$].DIST;
      END IF;
    END FOR;
    FOR EACH $obj \in$ object_distList DO /*refinement step*/
      IF (obj.DIST.complete() = true) THEN
        $d'_{prune} = 2 \cdot |S_Q| \cdot d_{TS}(S_Q, o)$;
        IF ($d'_{prune} < d_{prune}$) THEN
          result := obj.OID;
          $d_{prune} := d'_{prune}$;
        END IF;
        delete $obj$ from object_distList and prune it for further consideration;
      END IF;
    END FOR;
    FOR EACH $obj \in$ object_distList DO
      lb_dist := $d_1^{min}(S_Q, S_Y) + |S_Q| \cdot d_2^{min}(S_Q, S_Y)$;
      IF (lb_dist $\geq d_{prune}$) THEN
        delete $obj$ from object_distList and prune it for further consideration;
      END IF;
    END FOR;
  UNTIL (object_distList = empty);
  report result;
END
```

FIGURE 4.15: Threshold-based nearest-neighbor query algorithm.

4.6.1 Datasets and Methods

We used several real-world and synthetic datasets for our experimental evaluation. An overview of the datasets can be found in Chapter 3. For the efficiency evaluation we used subsets of the audio collection containing up to 700,000 time series objects with a length of up to 300 values per sequence. If not otherwise stated, the database size was set to 50,000 objects and the length of the objects was set to 50.

All experiments were performed on a workstation featuring a 1.8 GHz Opteron CPU and 8GB RAM. We used a disk with a transfer rate of 100 MB/s, a seek time of 3 ms and a latency delay of 2 ms. Performance is presented in terms of the elapsed time including IO and CPU-time.

4.6.2 Performance Results

We compared the efficiency of our proposed approach to a number of competing techniques. In the following we will denote our approach for answering threshold queries by 'R_{Par}'.

The first competing approach works on native time series. At query time the threshold-crossing time intervals are computed for the query threshold and afterwards the distance between the query time series and each database object can be derived. In the following this method will be denoted by 'Seq_{Nat}' as it corresponds to a sequential processing of the native data.

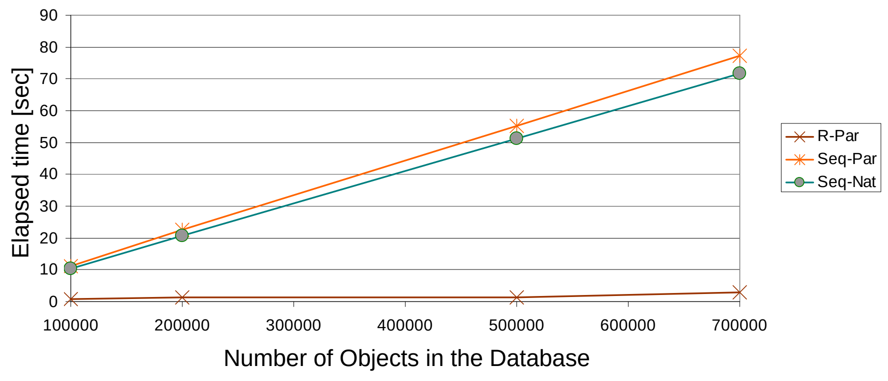
The second competitor works on the parameter space rather than on the native data. It assumes all time series objects have already been mapped to the parameter space. However, no index structure is used. As this storage leads to a sequential scan over the elements of the parameter space we will refer to this technique as the 'Seq_{Par}' method.

Furthermore we included a number of traditional similarity measures based on the following dimensionality reduction methods: Chebyshev Polynomials (*Cheb*) [CN04], Discrete Fourier Transformation (*DFT*) [AFS93], and Fast Map (*FM*) [FL95]. In particular, we implemented the algorithm proposed by Seidl and Kriegel in [SHP98] which adapts the GEMINI framework (cf. Section 2.5) for k-nearest-neighbor search. Since the applied dimensionality reduction techniques approximate the Euclidean space, they can only be used to accelerate similarity queries based on the Euclidean distance. They cannot be applied to threshold-based similarity search applications.

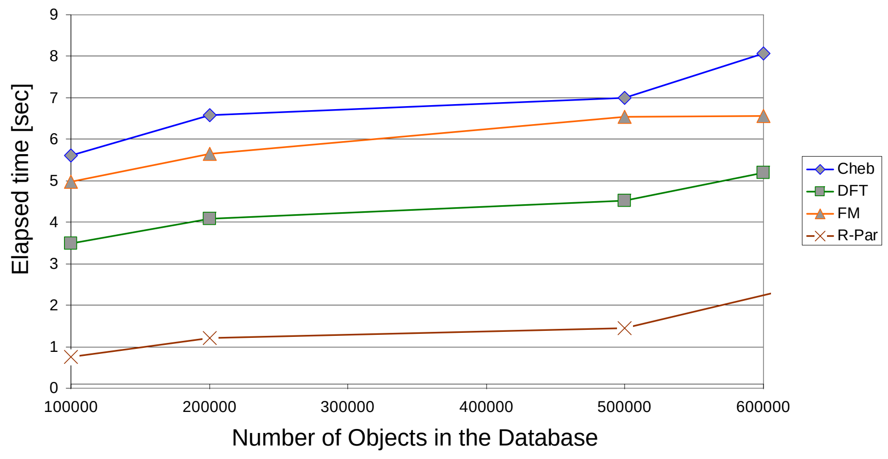
To obtain more reliable and significant results, in the following experiments we used 5 randomly chosen query objects. Furthermore, these query objects were used in conjunction with 5 different thresholds, so that we obtained 25 different threshold queries. The presented results are the average results of these queries.

We used the audio dataset and varied its size as well as the length of the time series. For the first experiment we varied the database size and set the length of the time series to a fixed value of 50 time slots. In Figure 4.16(a) the results of our approach compared to Seq_{Nat} and Seq_{Par} are given. The performance of Seq_{Nat} and Seq_{Par} decreases, while our approach can handle large amounts of data very well. The next experiment (cf. Figure 4.16(b)) compares our approach to the dimensionality reduction methods listed above. Although the scalability behavior of our approach is similar to that of the dimensionality reduction techniques, the absolute performance value of our approach is significantly better than that of the dimensionality reduction methods.

The next experiment explores the impact of the length of the query object and the time series in the database. The results are shown in Figure 4.17. Again, our technique outperforms Seq_{Nat} and Seq_{Par} (cf. Figure 4.17(a)), whose cost increase very fast due to the expensive distance computations. In contrast, our approach, like DFT and FM, scales well for larger time series objects. For small time series it even outperforms by far the three dimensionality reduction approaches as shown in Figure 4.17(b). If the length of the time series objects exceeds 200, then DFT and FM scale better then our approach. In contrast, Cheb scales relatively bad for larger time series. The reason is that the number of required Chebyshev coefficients has to be increased with the time series length for constant approximation quality.



(a)

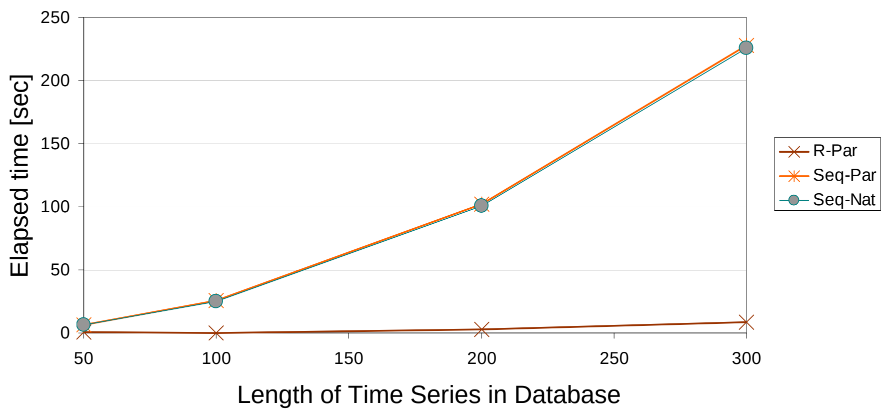


(b)

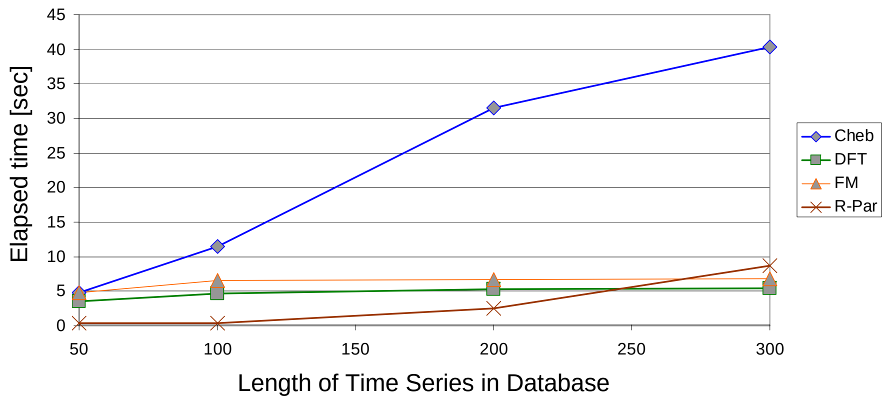
FIGURE 4.16: Scalability of the threshold-query algorithm with respect to the database size.

Obviously, the cardinality of our time series representations increases linear with the time series length.

In the next experiment, we analyzed the speed-up of the query process caused by our pruning strategy. We measured the number of result candidates considered in the filter step of our query algorithm, denoted by '*Filter*', and the number of objects which have to be refined finally, denoted by '*Refinement*'. Again,we compare our approach to the three dimensionality reduction methods Cheb, DFT, and FM. Figure 4.18(a) and Figure 4.18(b) show the results relatively to the database size and length of the time se-



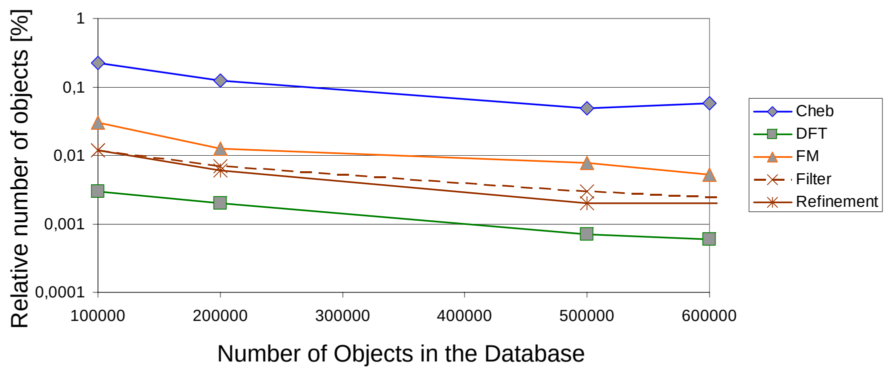
(a)



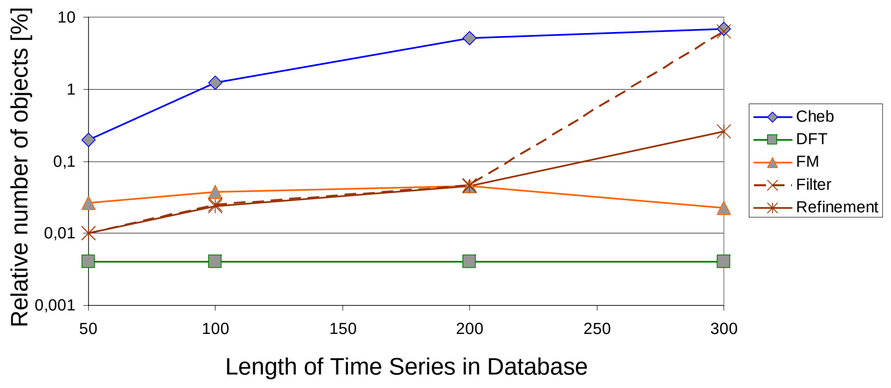
(b)

FIGURE 4.17: Scalability of the threshold-query algorithm with respect to the length of the time series.

ries objects. Generally, only a very small portion of the candidates has to be refined to calculate the result. Similar to the dimensionality reduction methods, our approach scales well for large databases. For small time series, our approach has a lightly better pruning power then Cheb and FM. We can observe that the pruning power of our approach decreases with increasing time series length. An interesting observation is that the number of candidates that have to be accessed in the filter step increases faster with larger time series than the number of finally refined candidates. Yet, for the audio dataset the DFT method shows the best results in terms of pruning power.



(a) Pruning power for varying database size.



(b) Pruning power for varying time series length.

FIGURE 4.18: Pruning Power of the threshold-based nearest-neighbor algorithm.

Furthermore, we examined the number of nearest-neighbor search iterations that were required for the query process for varying length of the time series and varying size of the database. We observed that the number of iterations was between 5 and 62. The number of iterations increases linear to the length of the time series and remains nearly constant with respect to the database size. Nevertheless, only a few iterations are required to report the result.

So, in terms of performance, our approach significantly outperforms Seq_{Nat} and Seq_{Par}. It is furthermore comparable to the above mentioned dimensionality reduction techniques Cheb, DFT, and FM. For time series of small and medium length our approach even outperforms these dimensionality reduc-

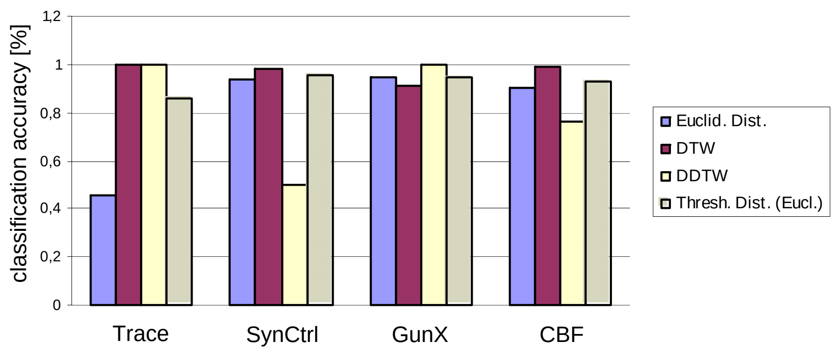


Figure 4.19: Comparison to Traditional Distance Measures.

tion techniques.

4.6.3 Threshold-Based Similarity Measure

In this section we present experimental results that underline the usefulness of our approach when applied to real-world datasets. We evaluated the quality of our similarity measure in terms of classification accuracy of a k-nearest-neighbor classifier, using a parameter setting of $k = 5$ and a 10-fold-cross validation.

Comparison to Traditional Distance Measures

In a first experiment (cf. Figure 4.19), we compared our approach to competing similarity measures which are traditionally used for time series data. In particular, we included the Euclidean distance (Eucld. Dist.), Dynamic Time Warping (DTW), and the Derivative Dynamic Time Warping (DDTW) [KP01] in our evaluation.

Our approach achieves good classification results for all considered datasets. For the Trace dataset the Euclidean distance achieves only an accuracy of about 45% while our approach achieves approximately 86%. On the GunX dataset our approach even outperforms the DTW distance measure.

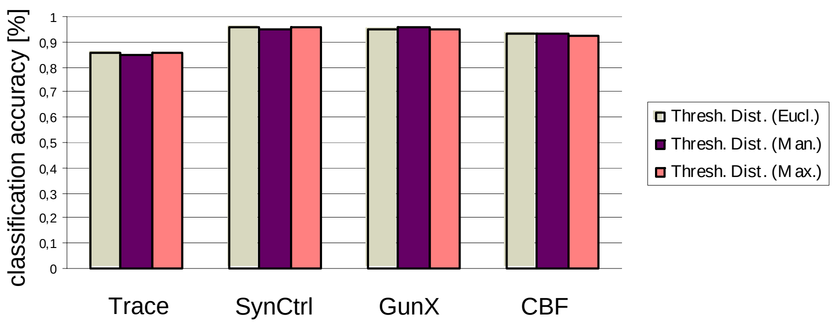


FIGURE 4.20: Comparison of Different Interval Similarity Distances.

Comparison of Different Similarity Distances for Time Intervals

First, we will examine different L_p-norms ($p = 1, 2, \infty$) applied to the interval-similarity distance measure d_{int}. Figure 4.20 shows the results of the classification accuracy achieved, respectively. As we have expected in Section 4.3.2, all three L_p-norms show a similar behavior in terms of classification accuracy.

Results on Scientific Datasets

Finally we performed 10-nearest neighbor threshold queries with randomly chosen query objects on the air pollution dataset. Interestingly, when we choose time series as query objects, that were derived from rural sensor stations representing particulate matter parameters (M_{10}), we obtained only time series representing the same parameters measured also at rural stations. This confirms that the pollution by particle components in the city differs considerably from the pollution in rural regions. A second interesting result was produced when we used M_{10} time series of working days as queries. The resulting time series were also derived from working days representing M_{10} values.

The results on the gene expression dataset were also very interesting. The task was to find the most similar gene with $\tau = 0$ to a given query gene. The intuition is to find a gene that is functionally related to the query gene. We executed several randomized queries to this dataset with $\tau = 0$ and evaluated the results with respect to the biological function using the SGD database[CBW$^+$97]. Indeed, we retrieved functionally related genes for most of the query genes. For example, for query gene CDC25 we obtained the gene CIK3. Both genes play an important role during the mitotic cell cycle. For the query gene DOM34 and MRPL17 we obtained two genes that are not yet labeled (ORF-names: YOR182C and YGR220C, respectively). However, all four genes are participating in the protein biosynthesis. In particular, threshold queries can be used to predict the function of genes whose biological role is not resolved yet.

4.7 Conclusions

To summarize the chapter, the results on the real-world datasets suggest the practical relevance of threshold queries for important real-world applications. In this chapter, we motivated and proposed a novel query type on time series databases called *threshold query*. Given a query object Q and a threshold τ, a *threshold query* returns time series in a database that exhibit the most similar threshold-crossing time interval sequence. The threshold-crossing time interval sequence of a time series represents the interval sequence of elements that have a value above the threshold τ. Furthermore, we presented a novel approach for managing time series data to efficiently support such threshold queries. Finally, we developed an efficient algorithm to answer *threshold queries* for arbitrary thresholds τ.

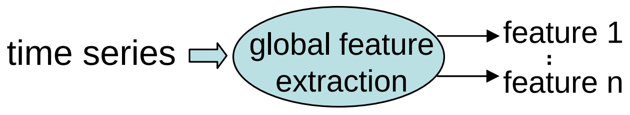
Chapter 5

Amplitude-Level-Based Similarity

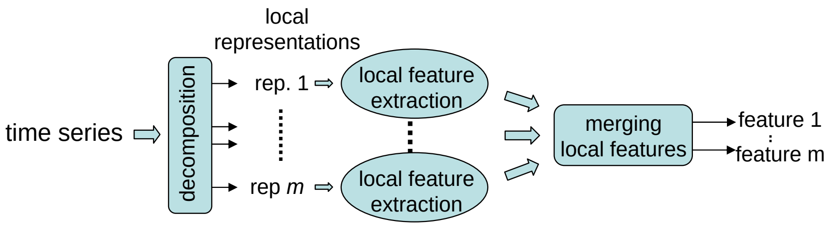
In the last chapter (see Chapter 4) we have defined a new similarity measure based on a user-given threshold. In this chapter we extend this idea and consider several thresholds. As described earlier in this thesis (see Section 2.3), the challenge for similarity search in complex data, and in particular in time series data is twofold. First, the adequate modeling of the similarity notion between time series is important for the accuracy of the search. This notion heavily depends on the application domain and the users involved in the search. Second, since time series are frequently very large, containing several thousands of values per sequence, the comparison of two time series can be very expensive, particularly when using distance measures that require the access to the raw time series data (i.e. the entire sequence of time series values). For example, for an audio sequence we can derive 300 features per second. Thus, a 3 minute audio sequence is represented by a time series of length 54,000.

Standard distance measures like the DTW or the Euclidean distance (see Section 2.4) usually compare quantitative information of time series. However, in particular for complex structured time series, features describing quantitative information are often quite susceptible to noise and outliers.

We propose a novel approach. First, a user can define a range of important or relevant amplitude values. Note this step is only optional, i.e. it is of



(a) Global



(b) Local

Figure 5.1: Global vs. local feature extraction.

course possible to consider the complete amplitude range as relevant. Then the specified range is scanned at a number of different threshold values. For each threshold value we extract a number of threshold-based features. For each kind of feature we obtain a course of feature values over the considered amplitude values. This sequence is called *feature sequence* and contains the actual information our similarity measure is based on. A feature sequence describes the qualitative characteristic of a time series with respect to a certain feature along the amplitude range. This is an important difference compared to traditional similarity measures or dimensionality reduction techniques, which aggregate information along the time dimension.

In order to be able to calculate distance values more efficiently, we compress the feature sequences by means of dimensionality reduction techniques. Finally, we combine different features to obtain a better representation for a given dataset. Even without compressing the feature sequences, the runtime complexity of our method is independent of the length of the time series which is important for very long multi-media time series.

Figure 5.1 depicts our novel approach of local feature extraction (cf. Figure 5.1(b)) in comparison to the traditional global feature extraction strategies [WSH06] (cf. Figure 5.1(a)). The traditional global approach extracts a set of n one-dimensional features representing the global characteristics of the time series. The resulting n features are used to build an n-dimensional feature vector and usually the Euclidean distance is used to measure the similarity between the derived features. In contrast, our approach is based on a decomposition of the complex structured time series into a reasonable set of more simply structured components which we call local representations. Each of the local representations corresponds to an amplitude value in the relevant amplitude range. Then we extract a set of local features of different types from these local representations. The main advantage of our strategy is that we dissect the complex feature extraction problem into a set of small subproblems which can be solved more easily. In order to reduce the computational cost of the similarity measure based on the resulting features, we can subsequently compress the results using standard dimensionality reduction techniques. In this work, we focus on one-dimensional time series. However, our approach can easily be adapted to the multi-dimensional case by extracting features for each dimension. In summary, our contributions are the following:

- We introduce a new similarity measure based on a range of relevant amplitude values.
- This similarity measure can be adjusted in a domain-specific way.
- We introduce several threshold-based features.
- We developed a framework for efficient and effective amplitude-wise comparison of time series.
- We show how prior knowledge about a dataset can be used to obtain similarity results of a higher quality.

The rest of the chapter is organized as follows. In Section 5.1, we survey related work. In Section 5.2, we present our feature extraction framework. A set of feature types reflecting the characteristics of time series is introduced in Section 5.3. Section 5.4 presents the experimental results.

5.1 Related Work

In Chapter 2 we give an overview of standard similarity measures. In this section, we additionally review existing feature extraction methods for time series.

For long time series usually structure level similarity measures based on global features or model parameter extraction are used [NAM01, DMN97, GS00, KLR04]. A similarity model for time series that considers the characteristics of the time series was proposed in [WSH06]. A set of global features including periodicity, self-similarity, skewness, and kurtosis are used to compute the similarity between the time series. Some of the features are generated from the raw time series data as well as from trend and seasonally adjusted time series. The authors focused on clustering as a special application of similarity search and showed that a small set of global features can be sufficient to achieve an adequate clustering quality. However, this approach is successful only as long as adequate features that reflect the time series characteristics can be identified. Unfortunately, long time series often feature very complex structures which cannot sufficiently be reflected by a single global feature, e.g. modeling the periodicity of a long time series with only one value may be too coarse in most cases.

In the multimedia area, publications about global features can be grouped in two main categories. The first category consists of approaches that calculate features in the so-called frequency domain. Well-known examples include Relative Spectral Predictive Linear Coding, Pitch [Sun02], Spectral Flux, Mel Frequency Cepstral Coefficients, Bark Frequency Cepstral Coefficients [LW01], and coefficients calculated by basic time-frequency transformations like DFT or DWT (cf. Section 2.5.3). The second category consists of techniques that extract features in the so-called time domain. Examples include Linear Predictive Coding coefficients [Tre82], Zero Crossing Rate Periodicity Histogram [Sun02], Sone and Short Time Energy [Pam04], Length of High Amplitude Sequence, Length of Low Amplitude Sequence, or Area of High Amplitude [MZB06].

In contrast to the existing features working in the time-based domain, the features proposed in this chapter are calculated over the whole amplitude spectrum. This fact allows us to capture time-domain properties along the whole available (or relevant) amplitude range. Moreover, we suggest an automatical method for the combination of the derived features which results in a significant improvement of effectiveness.

5.2 Considering Multiple Thresholds for Similarity Queries

As discussed above, traditional similarity measures and models for time series are often not appropriate to capture shape-based similarity of complex time series. Usually, these approaches apply variants of the Euclidean distance or DTW to quantitative representations of the time series, i.e. to the raw time series values or to features that are extracted from this quantitative representation.

We argue that it is more appropriate for shape-based similarity of time series to use a qualitative representation of the time series that models the shape characteristics of the time series. From such a qualitative representation of the time series, appropriate features can be extracted that capture the relevant characteristics of the time series. The simplest way for such a qualitative representation is the approach proposed in [RKBL05]. The time series is mapped to a sequence of intervals. Each interval represents the time slots at which the value of the time series is above a given amplitude level. In [RKBL05] the authors propose to use the mean value as the distinguishing amplitude level.

In order to compute a qualitative representation of a time series, we aggregate the time intervals corresponding to time slots where the amplitude values are above a given level. Thus, our qualitative representation of a time series consists of a sequence of time intervals. From this qualitative time series representation, features can be derived. As a result we end up with

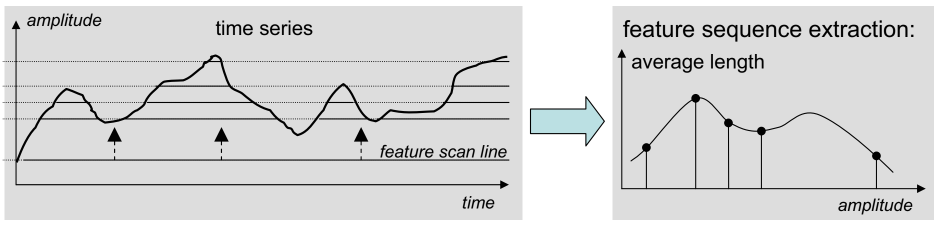


Figure 5.2: Amplitude-level-wise feature extraction.

one feature value that describes the complete time series qualitatively.

Obviously, using only one feature to describe a time series qualitatively will most likely be much too coarse, because the entire time series can usually not be described accurately by only one feature value. Rather, we should use all amplitude levels in order to capture any shape of the time series resulting in one feature for each amplitude value. However, this approach has two obvious drawbacks. First, the number of all amplitude values is infinite and so, the number of resulting features would also be infinite. Second, if a lot of amplitude values are considered, each local representation will be very similar to the next and to the previous local representations along the amplitude range. As a consequence, the derived features will be very similar too, and a lot of redundant information is stored.

To overcome this problem, we generate a sequence of feature values by scanning the amplitudes of any time series with a specific resolution. As a result we obtain a sequence of feature values, each value corresponding to the feature extracted for a given threshold. Obviously, the higher the resolution, the longer the feature sequence.

The main principle of our framework is depicted in Figure 5.2. The framework takes a time series as input and scans it at several amplitude values. This yields a feature sequence for each kind of extracted features. Our approach consists of two phases, the amplitude-level-wise feature extraction and the feature sequence compression. A detailed explanation of each phase is given in the next sections.

5.2.1 Amplitude-Level-Wise Feature Extraction

As for each considered amplitude value, we derive a feature value, we get a sequence of feature values, called *feature sequence*. This approach is illustrated in Figure 5.2. An input time series is scanned amplitude-level wise and based on the current threshold value, suitable features are extracted. For each kind of feature this technique yields a feature sequence. As we will outline in Section 5.2.2, it is also possible to apply dimensionality reduction techniques to these feature sequences. Finally we can combine the resulting feature vectors to improve the quality of similarity queries.

We propose a framework that extracts time series features in two steps: In a first step, we generate sequences of feature values by scanning the amplitudes of the corresponding time series with a reasonable high resolution. In order to improve the similarity search quality we extract several features from the interval sequences. As depicted in Figure 5.2 we use the feature scan line (fsl) to vertically scan the time series from bottom to top and retrieve at each (relevant) amplitude level τ a set of features called *Amplitude-Level Features* (ALFs). In Section 5.3 we give several examples for such Amplitude-Level features.

As a result, for each kind of considered ALF, we obtain a sequence $\langle(\tau_{min}, f_{\tau_{min}}), \ldots (\tau_{max}, f_{\tau_{max}})\rangle$, where τ_{min} denotes the global minimum of all amplitudes of all time series and τ_{max} denotes the corresponding global maximum of all amplitudes. f_τ denotes the ALF feature value for the current threshold τ. Note that it is of course also possible to specify a relevant amplitude range instead of considering the complete amplitude range. The resolution r of the amplitude scan (i.e. the length of the ALF sequence) is a user-defined parameter that influences the length of the resulting feature sequence as well as the accuracy of the representation. If we choose a high value for the resolution r, we will obviously obtain a more accurate description of the time series and may achieve better results. On the other hand, a high value for the resolution r results in larger space required to store the ALF sequences and in a lower query performance. In order to reduce the size of the extracted features and to decrease redundant information, we

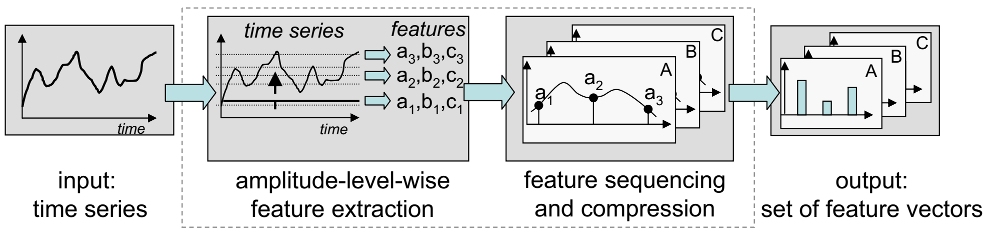


FIGURE 5.3: Feature extraction framework.

subsequently apply appropriate dimensionality reduction methods to reduce the large feature sequences to a smaller set of coefficients. These coefficients correspond to the feature vectors that are finally used to represent the time series and are used for the similarity search methods.

5.2.2 Feature Sequence Compression

Depending on the resolution r of the feature extraction method, the ALF sequences usually exhibit a more or less smooth shape. For a high resolution, the features extracted from adjacent amplitude thresholds do not differ very much. For this reason, common dimensionality reduction techniques for time series like DFT, PAA, or Chebyshev applied to the ALF sequences lead to shorter ALF sequences while accurately approximating the original ALF sequence. Finally, for each feature we generate a dimensionality reduced ALF sequence in the form of a vector which can be indexed by any spatial index structure. This strategy helps to solve performance issues while still yielding a high-quality similarity measure.

The principle of our framework is depicted in Figure 5.3. The framework takes a time series as input and produces a set of feature vectors as output. It consists of the two steps, the amplitude-level-wise feature extraction and the feature sequence compression. In particular, for a given time series and a given feature A we extract a sequence of local feature values a_1, a_2, and a_3. Subsequently, the generated ALF sequence is compressed by means of standard dimensionality reduction techniques resulting in a feature vector. This step is repeated for each feature extraction method so that finally a set of feature vectors is returned. This set is afterwards used to measure the similarity between the time series objects. Obviously, the final set of feature vectors and the dimension of each feature depends on the number and type of derived features, the resolution r, and the applied dimensionality reduction techniques. The length of the input time series however, has no influence on the dimensions of the resulting feature vectors. So the time complexity of a query is constant with respect to the length of the input time series. In the following section, we will present several high quality ALFs and discuss how to compute the similarity of the resulting set of feature vectors.

5.2.3 Feature Sequence Combination

As mentioned above, we generate a set of feature vectors for each time series. As the combination of different feature vectors usually improves the search quality we apply a combination approach similar to the techniques described in [BKS+04] and [KPS05]. In order to compute weights for the different representations, we use a labeled subset of the dataset as a training set. For each element of the training set, we perform a kNN query on the training set. The weighting is calculated based on the impurity of the kNN sphere. The more different classes are present in the set of nearest neighbors, the less suitable the current feature is assumed to answer the query. kNN sets with a low entropy, i.e. only few different class labels are considered more suitable. The weights are actually calculated by determining the average entropy for each representations. After having determined the weights, a standard combination method like *sum*, *product*, *min*, or *max* can be used to combine the distances according to the different feature representations.

5.3 Amplitude-Level Features

The number and type of adequate features depend on the application and on the data. In the following, we propose a selection of features that mainly reflect shape characteristics of time series, and thus, are suitable for a broad range of standard applications. Let us note that it is possible to design further amplitude-level features that might be more suitable for special applications. However, we will show in our experiments that even the basic features described in the following already yield a very accurate similarity model for shape-based similarity search in time series databases.

Let in the following $S_{\tau,X}$ denote the qualitative representation of time series X with respect to the amplitude level τ. $S_{\tau,X}$ actually is the threshold-crossing time interval sequence as defined in Section 4.3.1. $S_{\tau,X}$ consists of the intervals where the amplitude value of X is above τ.

5.3.1 Above Amplitude Level Quota

Maybe one of the most straightforward approaches to extract meaningful features that describe a time series X is to measure the fraction of time series values that exceed a given amplitude level τ. This ALF is called Above Amplitude Level Quota (ATQ)

$$ALF_{ATQ}(X,\tau) = \frac{1}{N}\left|\{x_i : x_i > \tau, i \in 1..N\}\right|$$

The resulting features values obviously range from 0 to 1, where ALF_{ATQ} decreases monotonously for increasing values of τ. A typical example for the resulting ALF curve is depicted in Figure 5.4. In this example, the resulting feature sequence exhibits a distinct shape and so, it is suitable to describe the original time series very well. In presence of a lot of noise in the original time series, the shape of the feature sequence becomes less and less distinct. Nonetheless, this feature extraction method allows for the detection of frequent values in the time series even when the time series suffers from heavy noise and at first sight appears to consist of random values only. This property is exemplified in Figure 5.5. Of course, all information regarding the time slots of the time series values is lost.

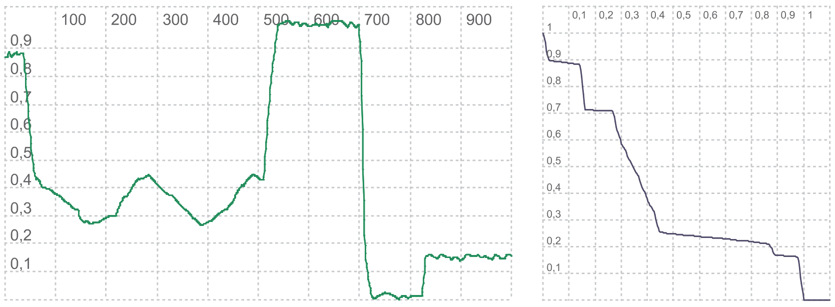


FIGURE 5.4: Sample time series and the corresponding feature sequence (ALF_{ATQ}).

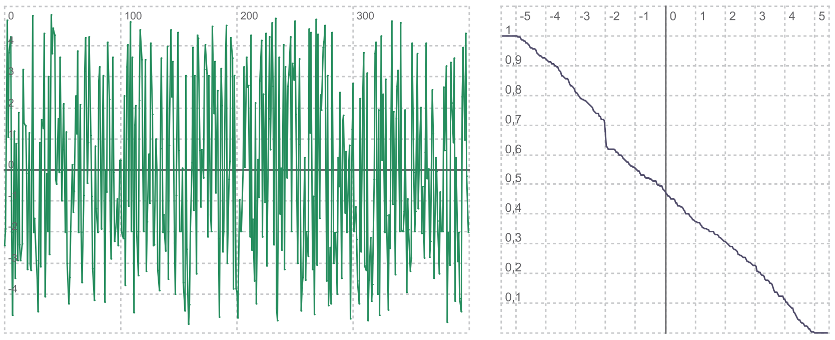


FIGURE 5.5: Sudden decrease in the ALF_{ATQ} feature sequence due to multiple occurrences of an identical value.

5.3.2 Threshold Interval Count

Another straightforward ALF is the number of amplitude level intervals generated for each amplitude level. We call this feature Threshold Interval Count (TIC). As the number of those intervals not only depends on the shape characteristics of a time series but also on the length N of the time series, N has to be considered for the calculation of the *Threshold Interval Count* feature $ALF_{TIC}(X, \tau)$. Normalizing the TIC feature with respect to the time series length, subsequently allows for the comparison of time series of different length. For a given time series X and an amplitude level τ, $ALF_{TIC}(X, \tau)$ is defined as:

$$ALF_{TIC}(X, \tau) = \frac{1}{N} |S_{\tau,X}|$$

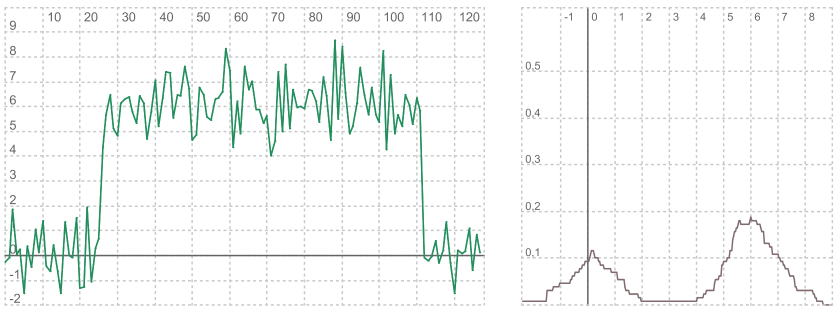


Figure 5.6: Sample time series and the corresponding feature sequence (ALF_{TIC}).

Unlike the *Above Amplitude Level Quota*, the range of occurring feature values cannot be tightly specified for all possible datasets. However, the maximal possible value equals $\frac{1}{2}$, although for most application domains the derived value of the *Threshold Interval Count* ALF is significantly smaller than the maximal value. So this maximum is not very well suited for operations like normalizing the feature values.

Noise has a stronger impact on this ALF feature sequence than on the feature sequence for ALF_{ATQ} as a lot of noise can lead to a huge number of intervals (cf. Figure 5.6). On the other hand, the ALF values differ more characteristically for varying amplitude levels at high frequency while lower values are observed in regions of a rather constant amplitude of the original time series. Similar to ALF_{ATQ} all chronological information is lost.

5.3.3 Threshold Interval Length (TIL)

In contrast to the ALF_{TIC}, TIL tries to capture more complex characteristics of the intervals than just their existence. An obvious choice is the average and the maximal length of all intervals for a given amplitude level τ and a time series X, formally

$$\begin{aligned} ALF_{maxTIL}(X,\tau) &= \max\{(u_j - l_j) : j \in S_{\tau,X}\} \\ ALF_{\varnothing TIL}(X,\tau) &= \frac{1}{|S_{\tau,X}|}\sum_{j=0}^{M}(u_j - l_j) \end{aligned}$$

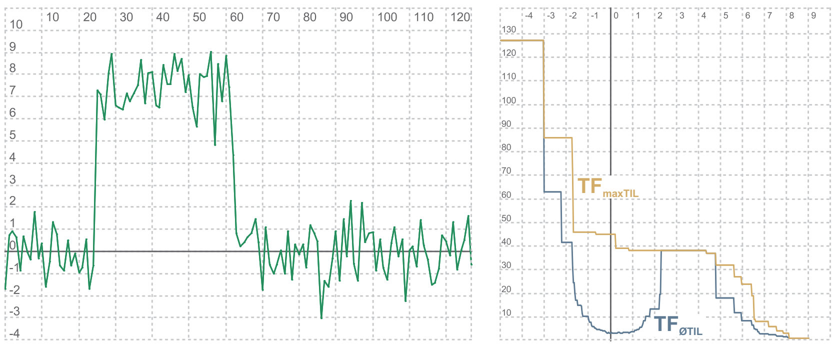


Figure 5.7: Sample time series and the corresponding feature sequence for ALF_{maxTIL} and $ALF_{\varnothing TIL}$.

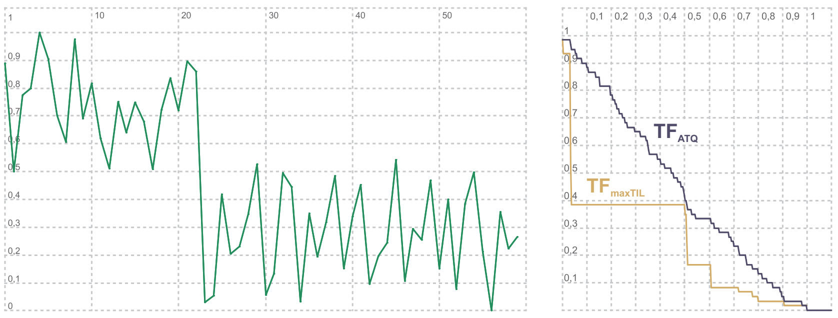


Figure 5.8: Robustness against noise of ALF_{maxTIL} compared to ALF_{ATQ}.

ALF_{maxTIL} and $ALF_{\varnothing TIL}$ show similar behavior in most cases. The former yields monotonously decreasing feature values and is more robust against noise. An example for both feature sequences is given in Figure 5.7. The contained information basically indicates at which amplitudes the time series consists of high or low frequent sections. For noisy data ALF_{maxTIL} is more effective than ALF_{ATQ}(see Figure 5.8). The feature sequence for ALF_{maxTIL} exhibits a very distinct pattern while ALF_{ATQ} is far less distinctive.

5.3.4 Threshold Interval Distance (TID)

A further reasonable feature is the distance between consecutive intervals. Generally, distances between intervals are ambiguous, so we have some options for the feature generation. We can build distances between the start points of the intervals only or between the end points only or we take both points into account. Here, we use the distance between the start point l_j of an interval and the start point l_{j+1} of the subsequent interval. We again consider both, the maximum and the average distance value:

$$\begin{aligned} ALF_{maxTID}(X,\tau) &= \max\{(l_{j+1}-l_j) : j \in S_{\tau,X}\} \\ ALF_{\varnothing TID}(X,\tau) &= \frac{1}{|S_{\tau,X}|-1} \sum_{j=1}^{|S_{\tau,X}|-1} (l_{j+1}-l_j) \end{aligned}$$

This feature differs from the others in being not invariant with respect to reflection along the time axis. It is adequate to separate periodical signals having different frequencies. Like ALF_{TIL}, ALF_{maxTID} is more robust against noise than $ALF_{\varnothing TID}$.

5.3.5 Threshold Crossing Angle (TXA)

This feature differs slightly from the previous as it does not take the interval sequences into account. Here, we consider the slopes of the time series that occur at the start and end points of the time intervals.

First, we define the slope angle angle(t_i) of a time series value $(x_i, t_i) \in X : i \in 2 \dots N$ as follows:

$$\text{angle}(t_i) := \arctan(x_i - x_{i-1}).$$

Based on this definition, we can define the features:

$$\begin{aligned} ALF_{absTXA}(X,\tau) &= \frac{1}{N\pi} \sum_{j=1}^{|S_{\tau,X}|} (|\text{angle}(l_j)| + |\text{angle}(u_j)|) * (u_j - l_j) \\ ALF_{diffTXA}(X,\tau) &= 0{,}5 + \frac{1}{N\pi} \sum_{j=1}^{|S_{\tau,X}|} (\text{angle}(l_j) + \text{angle}(u_j)) * (u_j - l_j) \end{aligned}$$

We weight the slope angles according to the corresponding interval length. The factor $\frac{1}{N\pi}$ as well as the constant value 0.5 are used to normalize the results to the range $(0, 1)$ and are necessary to compare time series of different lengths. This feature is mainly suitable to distinguish between time series

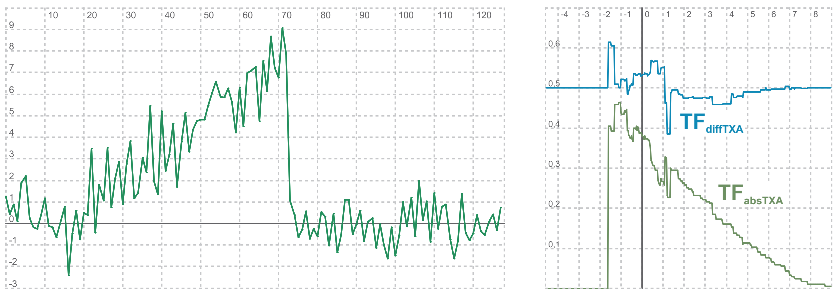


FIGURE 5.9: Sample time series and the corresponding feature sequence for ALF_{absTXA} and $ALF_{diffTXA}$.

with different rates of change. Unfortunately, the determination of such kind of patterns can be easily perturbed by noise. As a consequence, the quality of the similarity measures based on this feature mainly depends on the intensity of the noise in a dataset. An example for both feature sequences is depicted in Figure 5.9.

5.3.6 Threshold Balance (TB)

The last feature incorporates the temporal behavior of the time series by considering the temporal distribution of the amplitude values that are above the corresponding amplitude threshold. First, we define an auxiliary function

$$\text{above}_\tau(x_i) := \begin{cases} 1 & \text{if } x_i > \tau \\ 0 & \text{else} \end{cases}.$$

By means of this function, we can define the Threshold-Balance feature ALF_{TB} that aggregates those values of the time series which are above the amplitude threshold. An example for this feature is depicted in Figure 5.10.

$$ALF_{TB}(X,\tau) = \frac{1}{N^2}\sum_{i=1}^{N}\left(i - \frac{N}{2}\right)\text{above}_\tau(x_i).$$

Each of the presented features covers specific characteristics of a time series. Naturally, depending on the application, the full power of the features can be achieved if we use combinations of them.

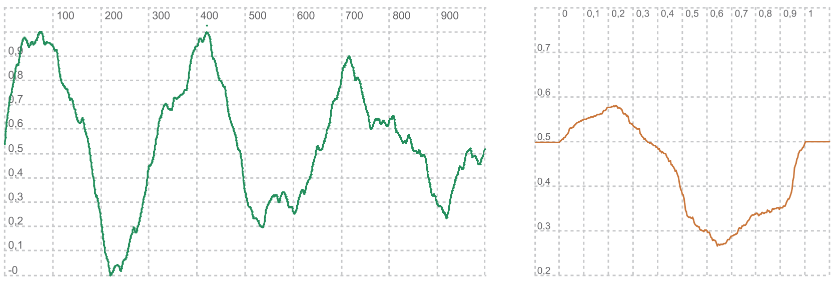


FIGURE 5.10: Sample time series and the corresponding feature sequence for ALF_{TB}.

5.4 Evaluation

In this section we present the results of the experimental evaluation of our amplitude-level-based similarity measure.

5.4.1 Datasets and Methods

We used four datasets from the UCR Time Series Data Mining Archive [KF02] as described in Chapter 3. The first dataset is the SynthCtrl dataset and will be referred to as "DS1" in the following. "DS2" actually is the GunX dataset. The Trace dataset will be labeled by "DS3", and "DS4" will be used to indicate the use of the Leaf dataset.

Whenever we compared the effectiveness of our new similarity measure, we performed precision-recall experiments. For a given dataset, we used each time series instance as a query object and ranked all other time series according to their similarity value to the current query. Then we iterated through the ranked list until we reached a certain recall level for the class of the query objects, i.e. until a certain portion of the class of the query was retrieved. The precision was then measured as the portion of objects of the same class as the query objects relative to the number of all objects retrieved so far. Obviously, the precision is larger or equal to 0 and smaller or equal to 1. Higher precision value indicate a better similarity measure. For a given dataset we averaged all calculated precision values over all queries and over all recall values in order to calculate a single average precision value.

We compared our approach to a number of competitors. First, we calculated the Euclidean distance on the raw time series, i.e. no feature sequences or dimensionality reduction techniques were applied. The second competing technique we included in our evaluation was the DTW [BC94] distance on the raw time series. Furthermore we compared our technique to the global feature extraction approach from [WSH06]. For this approach we compared an unweighted and a weighted version for the combination of the single global features the method consists of. We used the same combination method as for the combination of our amplitude-level features (see Section 5.2.3).

Whenever we calculated weights in order to combine single features, we used 20% of the current dataset as a training set in order to adjust the weights.

5.4.2 Experimental Results

Feature Quality

In a first set of experiments we evaluated the quality of the proposed amplitude-level features as introduced in Section 5.3. The results for the different datasets are given in Figure 5.11 for DS1, in Figure 5.12 for DS2, in Figure 5.13 for DS3, and in Figure 5.14 for DS4. As the purpose of this set of experiments was to evaluate the suitability of the single amplitude-level features, we did not compress the feature sequences but rather used the complete feature sequence to compute distance values. In the above mentioned figures the yellow bar indicates the unweighted combination of the single ALFs. The single ALFs themselves are indicated by blue bars. The Euclidean distance, the DTW, and the global features are presented in green color. Note that we used an unweighted combination for the global features as well, as we assumed an unsupervised setting for the first experiments.

As expected, the average precision values for the single ALFs vary significantly over the analyzed datasets. The datasets have different characteristics

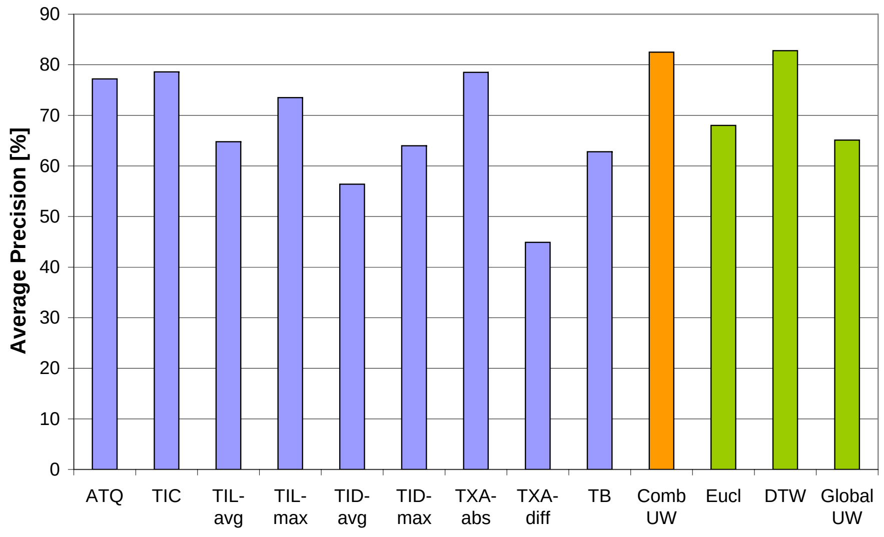


Figure 5.11: Average precision of single ALFs (DS1).

and furthermore, the different classes of different datasets may be distinguishable based on different properties. The nice observation we were able to make is that although we have not used any knowledge about the underlying classification system, the unsupervised combination of all ALFs yields an average precision value which is higher or comparable to the best competitor, the DTW. Note that the average precision of the unweighted combination of the ALFs is higher than the best single ALF for a given dataset on three out of four datasets.

In the light of this observation, it is interesting to compare the runtime of the different approaches. In Figure 5.15 we depicted the average runtime per query for the first set of experiments as described above. DTW, the only competitor that is comparable to our approach in terms of average precision values, suffers from a significantly higher runtime. This is due to the quadratic complexity in the length of the time series of the DTW. In contrast, the runtime complexity of our approach is independent of the length of input time series. The runtime of our approach is determined by the length of the feature sequences, as only the feature sequences (or their compressed representations) have to be compared instead of the usually longer raw time series.

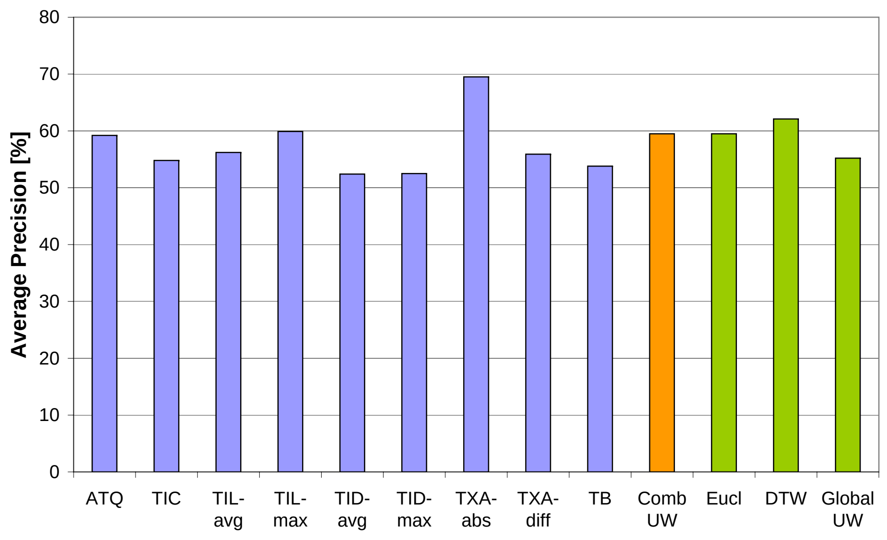


FIGURE 5.12: Average precision of single ALFs (DS2).

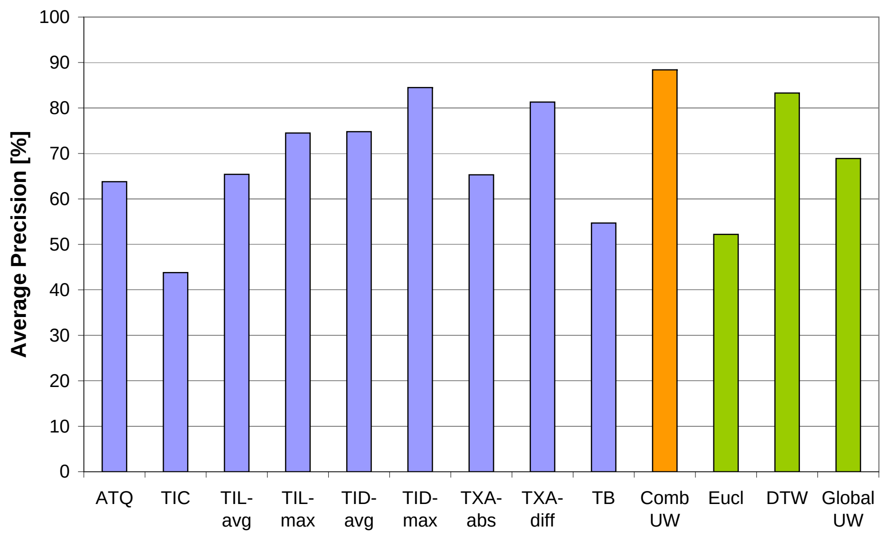


FIGURE 5.13: Average precision of single ALFs (DS3).

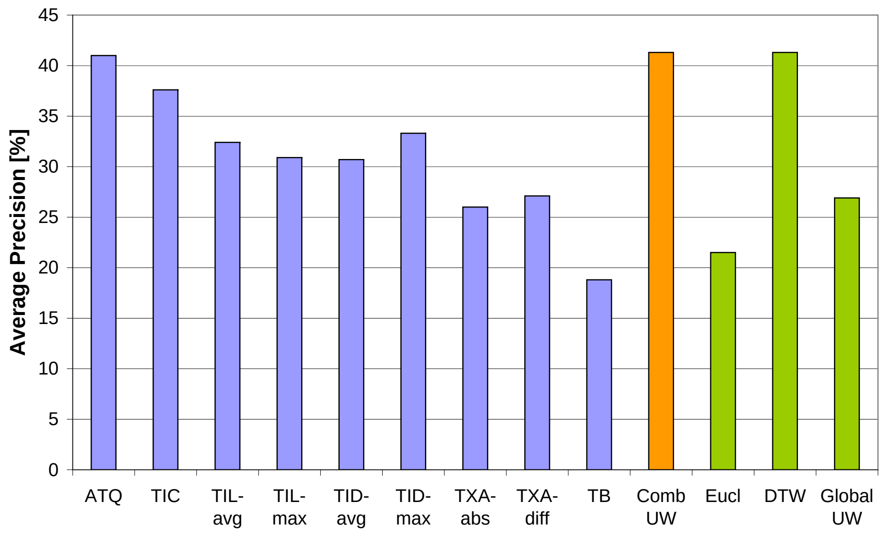


Figure 5.14: Average precision of single ALFs (DS4).

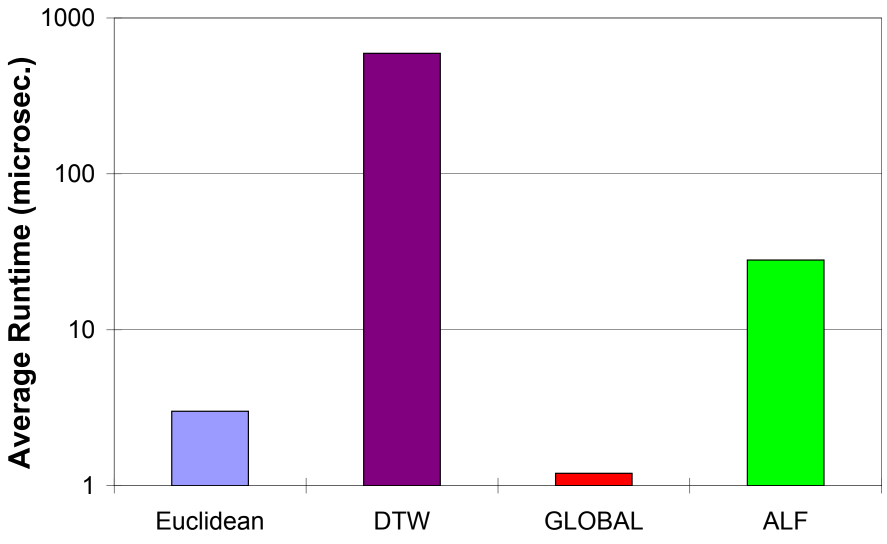


Figure 5.15: Average runtime per query.

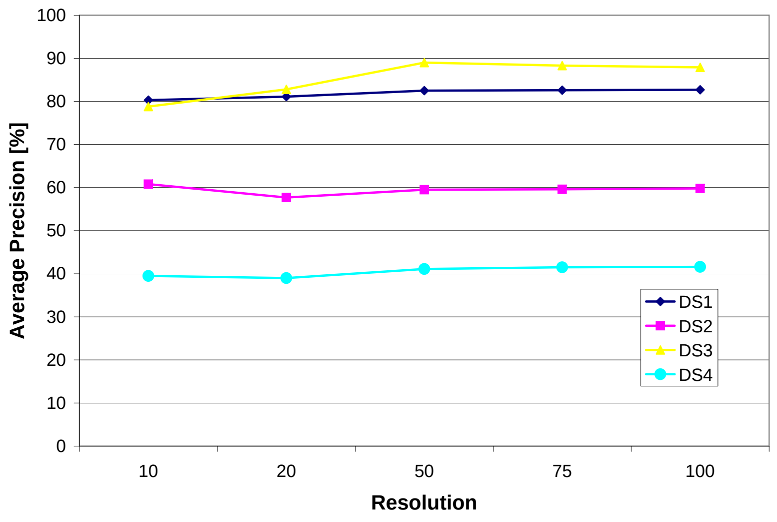


FIGURE 5.16: Average precision values of unweighted ALF combination for different resolutions.

This lead us to the question about the importance of the length of the feature sequences. The resolution that is used to scan over the given amplitude range might be a crucial parameter. So we calculated average precision values for all datasets based on an unweighted combination of uncompressed feature sequences for varying resolutions. The results of this experiment can be found in Figure 5.16. We observed that relatively few amplitude level are sufficient to describe even long time series by means of feature sequences. Furthermore, our approach is quite robust with respect to the choice of the resolution. Following the results of this experiment, for the remaining experiments (unless stated otherwise) we kept the resolution fixed at 50 amplitude values distributed equally over the completed range of possible amplitude values for a given dataset.

Feature Sequence Compression

In this section we present experiments analyzing the impact of the compression of the feature sequences. In Figure 5.17 the average precision values of an unweighted combination of all described ALFs are depicted for all datasets. The Figure compares the results for the uncompressed (raw)

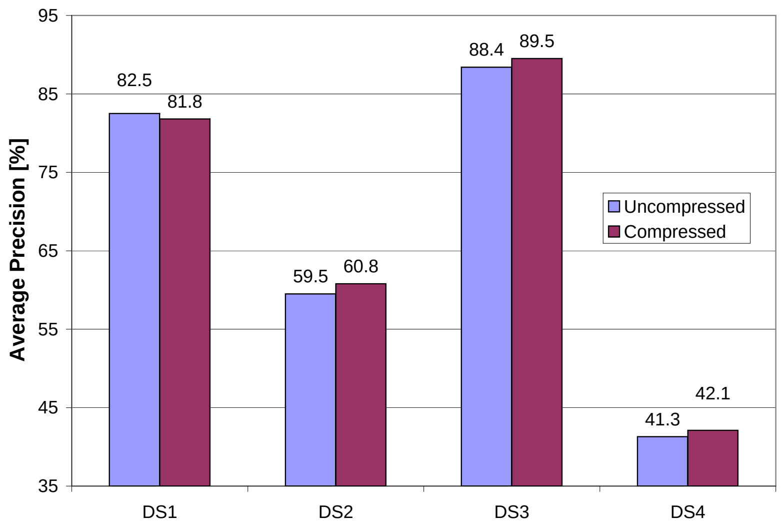


Figure 5.17: Impact of Compression of Features Sequences.

feature sequences to the results observed when using 15 PAA coefficients in order to efficiently describe each feature sequence. On all dataset except on DS1 we can observe an increase in the average precision value. This result may be surprising at first glance, as one might expect the feature sequences to be less precisely represented after a dimensionality reduction technique was applied. However we observed that in our case the application of such a dimensionality reduction technique can help to level out unimportant details of the raw time series captured in the exact representation of the feature sequences.

Note that after the feature sequences have been reduced to a few coefficients, the average runtime of our approach is significantly decreased compared to the value in Figure 5.15 as only a few coefficients have to be compared at query time instead of the complete feature sequences.

In a further experiment we analyzed the impact of different techniques fo dimensionality reduction. In Figure 5.18 we present the average precision values for several single ALFs on DS2 for two different dimensionality reduction techniques (DFT and PAA). As expected, different reduction techniques are more or less suitable for different feature sequences, and sometimes the

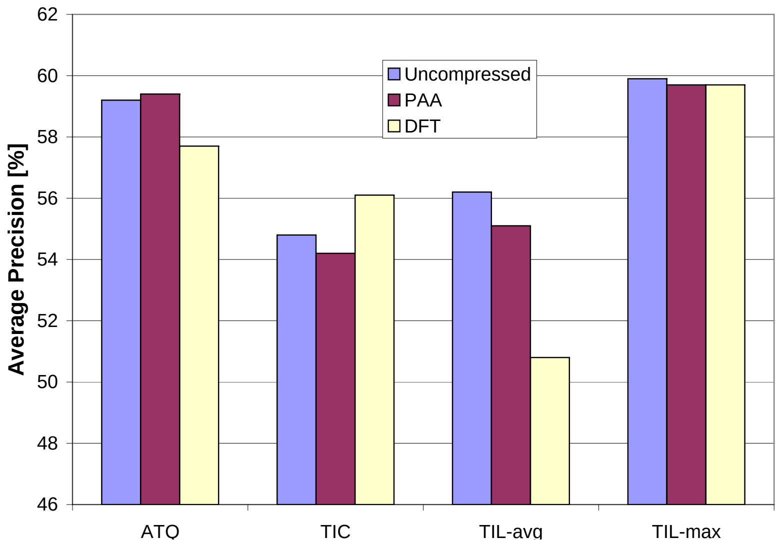


FIGURE 5.18: Impact of compression techniques on different features sequences (DS2).

best representation is the uncompressed one. This observation suggests a potential improvement of our method. Instead of using the same compression techniques for all feature sequences, it may be beneficial to develop a method which automatically chooses the most suitable compression technique for a given dataset and for each ALF feature sequence.

As the compression step of our approach has a significant impact on the average precision results, the next experiment explores the impact of the degree of the compression, i.e. how many coefficients are used to describe each feature sequence. In Figure 5.19 we present the average precision results for a varying number of PAA coefficients that were used to approximate each of the ALF feature sequences. For the range from 5 to 17 coefficients we observed quite stable precision results. More remarkably, the same range was most suitable for a resolution twice as hight as in the first run of the experiment. So, even if the feature sequences were 100 instead of 50 values long, a PAA setting between 5 and 17 coefficients was optimal to represent the ALFs of DS2. Of course the number of coefficients depends on the chosen method for the compression step and the dataset. However, we observed a broad range of well suited values for the number of coefficients for other

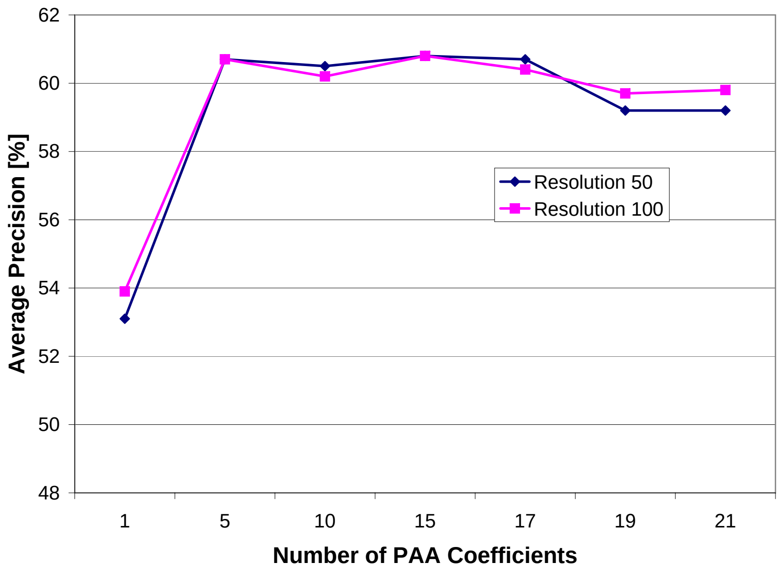


FIGURE 5.19: Impact of the degree of compression of feature sequences (DS2).

datasets and other compression techniques as well.

Feature Combination

After the compression of feature sequences, the next step in our approach is the combination of compressed or uncompressed feature sequences. In the experiments described so far, we used an unweighted combination of the single ALF sequences. The unweighted approach can always be used, especially in case no labeled training data is available. In this section we assume that a subset of the dataset is labeled in advance, so that for each dataset the most suitable weighting of feature sequences can be learned. This may even lead to the near exclusion of a certain feature if it is of no relevance for distinguishing different classes. In Figure 5.20 we depicted the average precision results for uncompressed feature sequences for all four datasets. As can be seen in this figure, the average precision is increased for all four datasets when the weights for each ALF are adjusted according to the specific dataset. The same can be observed for compressed features sequences. The

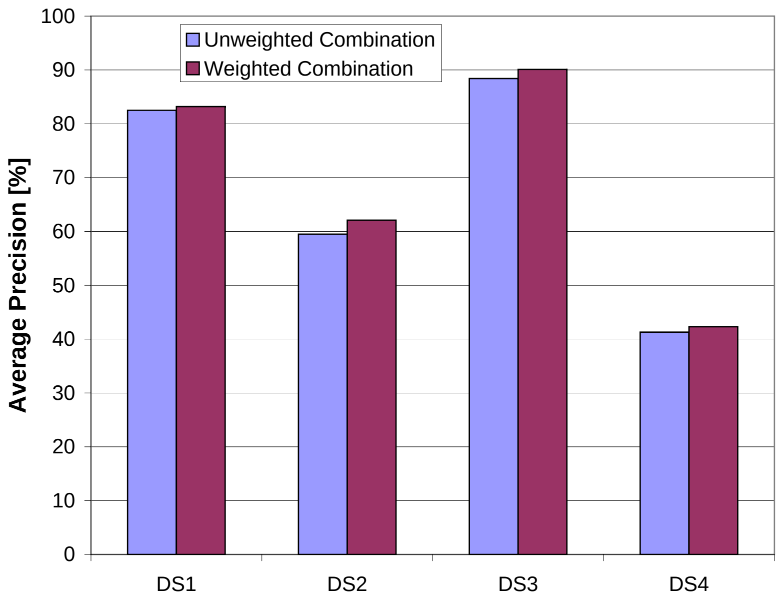


FIGURE 5.20: Weighted combination increases the average precision (uncompressed feature sequences).

corresponding results are given in Figure 5.21.

In order to give a final overview of our approach compared to other techniques, Figure 5.22 lists the result for DTW, the Euclidean distance, the weighted combination of global features, and our approach based on compressed and supervised combined feature sequences. Our approach clearly outperforms the global-feature approach and the Euclidean distance. For three out of four datasets it achieves significantly better precision results and on DS1 it achieves the same precision result as the DTW distance. At the same time, the runtime is constant while the runtime of the DTW is quadratic in the length of the compared time series.

The last experiment was performed to demonstrate a further benefit of our approach. The amplitude-level based similarity allows to incorporate domain knowledge into the similarity measure by restricting the range of amplitude levels considered for the feature sequence extraction. As shown above, even if the complete amplitude range of a dataset is used to extract feature sequences, very competitive results can be obtained. As shown in

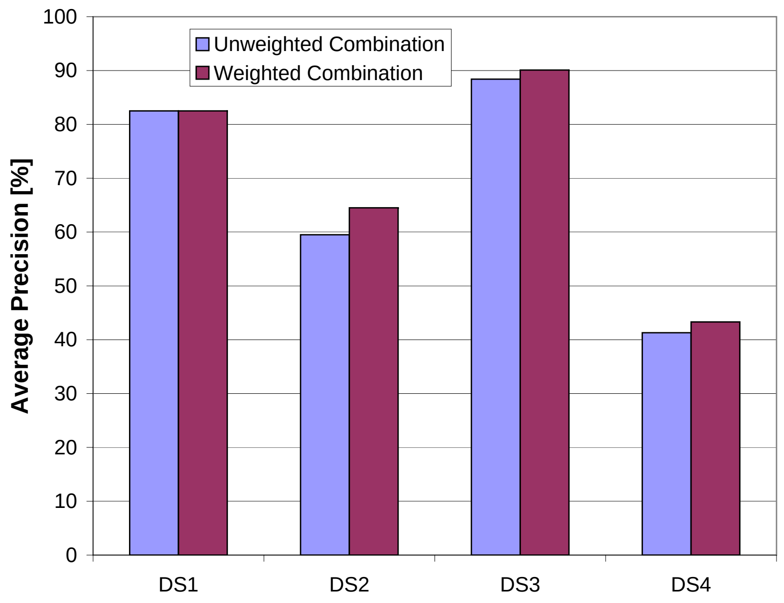


FIGURE 5.21: Weighted combination increases the average precision (compressed feature sequences).

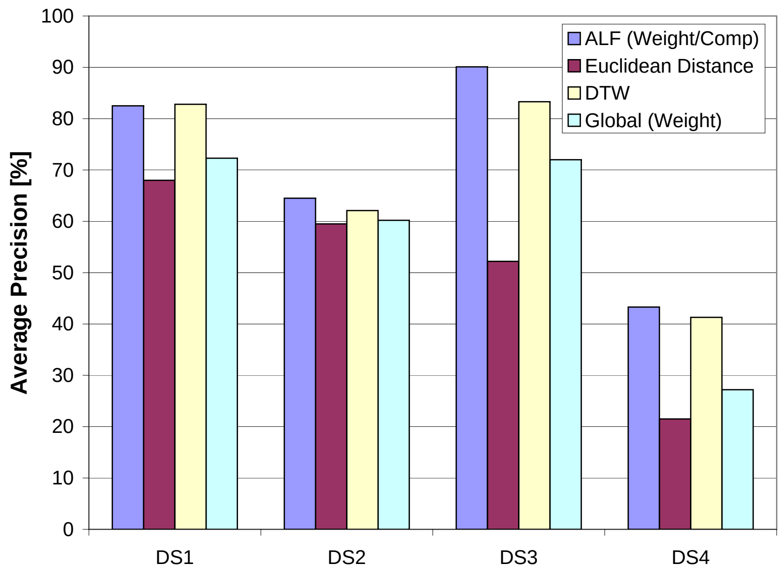


FIGURE 5.22: Average precision of competitors compared to compressed feature sequences in concert with a weighted distance combination.

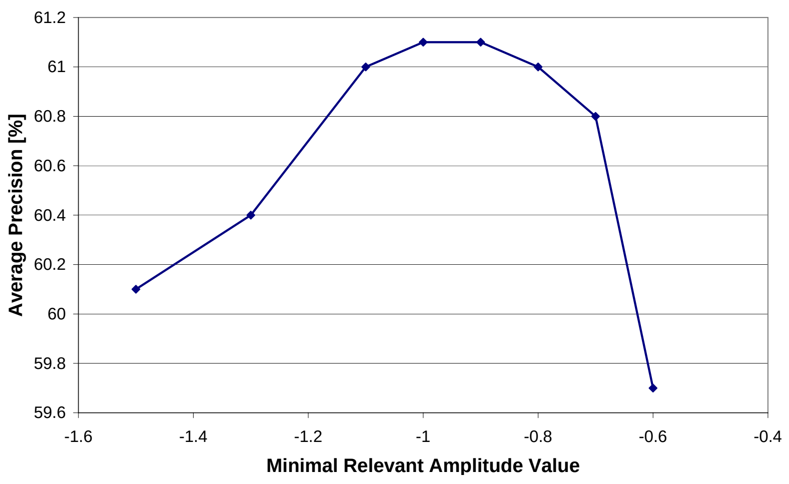


FIGURE 5.23: A restricted relevant amplitude range can yield higher precision values (DS2).

Figure 5.23, these results can be increased even further. Instead of considering the complete amplitude range, we set the maximal amplitude value to +2 for this experiment and varied the minimal amplitude value. The experiment shows that the precision values indeed change for different relevant amplitude ranges. Using our approach, it is even possible to specify several relevant amplitude ranges.

5.5 Conclusions

In this chapter we proposed a new framework for generating high quality Amplitude-Level Features (ALF) from time series. An advantage when using ALFs for similarity search is that the runtime is independent of the length of the time series. Thus, ALFs are adequate even for long sequences as occurring frequently in multimedia applications. We furthermore introduced several ALFs that are able to describe the characteristic properties of a time series. We also proposed a method to combine several feature representations. We showed in our experimental evaluation that our proposed technique outperforms traditional similarity search methods in terms of accuracy. In addition, our approach significantly outperforms the only competitor that achieves roughly similar accuracy in terms of runtime. We showed that the compressed feature sequences often yield even better search results than the uncompressed feature sequences. The number of features after the compression is quite low and so it is possible to use an index structure without running into the curse of dimensionality. Finally, we showed how our similarity measure can be adjusted by a domain expert, in order to restrict the considered portions of a time series to a relevant range of amplitude values.

Chapter 6

Interval-Focused Similarity

In this chapter, we introduce yet another similarity measure for time series. Existing work usually focus either on a full comparison, i.e. the entire time series are compared by using an appropriate distance function, or on subsequence matching, i.e. all time series objects that match a subsequence are retrieved. However, in many applications, only predefined parts of the time series are relevant for a similarity query rather than the entire time series data. The time intervals of these predefined parts are fixed for all time series. Usually, these parts are specified by the user depending on the analysis focus and change from query to query. We call such type of queries, where only a small part of the entire time series is relevant *interval-focused similarity queries*. Obviously, interval-focused similarity is a generalization of a full comparison of the time series. On the other hand, the subsequence matching approach is orthogonal to interval-focused similarity. The task in subsequence matching is to find a suitable subsequence that best fits to another subsequence, even if the time slots do not correspond to each other (see Section 2.4.3). For interval-focused similarity search, the interval relevant for the query is fixed for all time series objects. A comparison of complete matching, subsequence matching, and interval-focused matching is given in Figure 6.1.

The notion of interval-focused similarity queries is a useful concept in many applications.

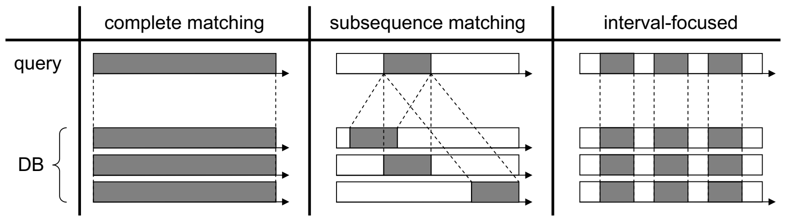


Figure 6.1: Different approaches for time series analysis.

In stock market analysis, the behavior of the stock prices is examined with respect to a given set of events such as political crises or seasonal phenomena. The time series are compared using interval-focused similarity queries that take only some relevant time periods into account, for example a certain time period after some event. The analysis of the annual balances of a company is usually also focused on specific time intervals like months or quarters.

In environmental research, important parameters like the concentration of ozone, the temperature, or the precipitation is usually measured over long time periods at various locations. While this kind of data is suitable for long-term analysis, for short-term observations, a smaller time frame might be of interest. For example, an interval-focused query may be based on the values for a certain week or a several days. Note that these relevant intervals do not necessarily have to be adjacent to each other.

In behavioral research, brain waves of animals are recorded throughout a given time period, for example a complete day. Researchers often want to compare the brain waves of different individuals during a significant time interval. So, the considered interval could for example correspond to the feeding phase rather than to the complete day. Obviously, in all these applications, the focus of the analysis task frequently changes from time to time and is not known in advance.

In summary, our contributions in this chapter are the following:

- We formalize the novel notion of interval-focused similarity queries, an important generalization of comparing complete time series.
- We propose a novel method to efficiently support interval-focused dis-

tance range and k-nearest neighbor queries that implements a filter-refinement architecture.

- Furthermore, we show how the proposed interval representation approximating the time series can be efficiently accessed using an index structure.

The remainder is organized as follows. We discuss related work in Section 6.1. The novel interval-focused similarity measure is formalized in Section 6.2. In Section 6.3 we introduce an interval-based representation of time series. We further show how these representations can be managed efficiently in order to upper and lower bound the distance between time series objects. Based on these bounds we present a filter-refinement architecture to support interval-focused similarity queries efficiently. We discuss two methods for the generation of interval representations of time series in Section 6.3.3. Section 6.5 provides an experimental evaluation of our proposed methods.

6.1 Related Work

The similarity between two time series is usually measured by an appropriate distance function as outlined in Chapter 2. Existing approaches focus either on complete matching of the query time series with the database objects, or on subsequence matching.

Complete matching approaches consider the complete time course using any of the above mentioned distance measures An example is depicted in Figure 6.1 (left). Since the length of a time series is usually very large, the analysis of time series data is limited by the well-known *curse of dimensionality.* Usually, dimensionality reduction methods are applied as described in Section 2.5.3.

The above-mentioned dimensionality reduction techniques are not designed for interval-focused similarity queries. Often, they use the complete time series to extract coefficients, like in case of the DFT. Consequently, the frequency domain coefficients can not be used to calculate distance approximations in the time domain, if the time domain has changed (by restricting the similarity computation to a relevant set of intervals).

Subsequence matching approaches usually try to match a query subsequence to subsequences of the database objects as depicted in Figure 6.1 (middle). A subsequence matching problem can be transferred to a complete matching problem by moving a sliding window over each time series object in the database and materializing the corresponding subsequence. If the length of the query subsequence changes, a new sliding window has to be moved over each database time series again. Subsequence matching is orthogonal to interval-focused similarity. In interval-focused similarity, the time slot relevant for the matching is fixed. Two time series are not considered similar even if they have a similar subsequence but at different time intervals. In addition, the concept of interval-focused similarity allows to specify multiple relevant time intervals of different length.

6.2 Interval-Focused Queries

In this section we formally introduce the concept of interval-focused queries.

6.2.1 Time Interval Sequence

Let $\mathcal{D}$ denote a database of time series. Let $X \in \mathcal{D}, X = \langle (x_1, t_1), .., (x_N, t_N) \rangle$ be a time series as defined in Definition 2.1 of length N. Let MAX be the maximal amplitude value over all time series in $\mathcal{D}$ and let MIN be the minimal amplitude value occurring in $\mathcal{D}$, i.e.

$$MAX = \max_{X \in \mathcal{D}} \left(\max_{i=1...N} (x_i) \right) \qquad MIN = \min_{X \in \mathcal{D}} \left(\min_{i=1...N} (x_i) \right)$$

Definition 6.1 (Time Interval Sequence).
Let the time interval *$I = (l, u) \in T \times T$ be a pair of time slots, where $0 \le l \le u \le N$. l denotes the starting point and u denotes the end point of I. Given a time series $X \in \mathcal{D}$ and an interval I, the* time interval sequence

of X corresponding to I *is a time series of length* $(u - l) + 1$ *consisting of the values of* X *between the start and the end time slot of* I*.*

$$X_I = \langle (x_l, t_l), \dots, (x_u, t_u) \rangle$$

6.2.2 Similarity Model for Time Interval Sequences

In this section we extend the standard Minkowski distance measures as defined in Definition 2.5 to interval-focused time series. As discussed above, interval-focused similarity specifies a given part of the time series as relevant, whereas the remaining part of the time series is considered irrelevant. The relevant part may change from query to query. In case of a single relevant interval the interval-focused similarity is defined as follows.

Definition 6.2 (Interval-Focused Similarity for a Single Interval).
Let X *and* Y *be two time series. Let* $I = (l, u)$ *be a time interval. Then the* L_p*-norm between* X *and* Y *with respect to* I *is defined as*

$$L_p^I(X, Y) = \sqrt[p]{\sum_{i=l}^{u} |x_i - y_i|^p}.$$

In order to calculate interval-focused similarity values based on several relevant stretches of time, we extend the last definition to a number of relevant intervals.

Definition 6.3 (Interval-Focused Similarity).
Let $\mathcal{I}$ *be a set of time intervals. Then* $L_p^{\mathcal{I}}(X, Y)$*, the* L_p*-norm between* X *and* Y *with respect to* $\mathcal{I}$ *is defined as*

$$L_p^{\mathcal{I}}(X, Y) = \sqrt[p]{\sum_{I \in \mathcal{I}} L_p^I(X, Y)^p}.$$

Note that the intervals $I \in \mathcal{I}$ can be of varying length and thus, the influence of each interval on the complete sum may be different. In some applications, it may be more appropriate to weight the intervals, such that the contribution to the overall distance of each interval is similar. This can be easily achieved by multiplying a weighting factor w_I to each summand.

$$L_p^{\mathcal{I}}(X,Y) = \sqrt[p]{\sum_{I \in \mathcal{I}} w_i L_p^I(X,Y)^p}.$$

6.2.3 Similarity Queries for Time Interval Sequences

Based on the distance measure introduced in the previous section, we can now extend the two most widely used similarity queries, the *distance range query* and the *k-nearest-neighbor query.* According to Definition 2.6, the distance range query retrieves all objects of a database whose distance to a given query object Q is smaller or equal to a specified distance value ε. The *k-nearest neighbor query* (kNN query) (see Definition 2.7) reports the k most similar objects to Q.

Definition 6.4 (Interval-Focused ε-Range Query).
Let $\mathcal{D}$ be a database of time series objects. The interval-focused ε-range query *consists of a query time series Q, a distance parameter $\varepsilon \in \mathbb{R}_0^+$, and a set of relevant time intervals $\mathcal{I}$. The* interval-focused ε-range query *retrieves the set $IQ_\varepsilon^{range}(Q,\mathcal{I}) \subseteq \mathcal{D}$ such that*

$$\forall X \in IQ_\varepsilon^{range}(Q,\mathcal{I}) : L_p^{\mathcal{I}}(X,Y) \leq \varepsilon$$

Analogously we extend the definition of the kNN query.

Definition 6.5 (Interval-Focused k-Nearest Neighbor Query).
Let $\mathcal{D}$ be a database of time series objects. The interval-focused k-nearest neighbor query *consists of a query time series Q, a parameter $k \in \mathbb{N}^+$, and a set of relevant time intervals $\mathcal{I}$. The* interval-focused k-nearest neighbor query *retrieves the smallest set $IQ_k^{NN}(Q,\mathcal{I}) \subseteq \mathcal{D}$ that contains at least k elements such that*

$$\forall X \in IQ_k^{NN}(Q,\mathcal{I}), \forall Y \in \mathcal{D} \setminus IQ_k^{NN}(Q,\mathcal{I}) :$$

$$L_p^{\mathcal{I}}(Q,X) \leq L_p^{\mathcal{I}}(Q,Y).$$

After we have formalized the notion of interval-focused similarity queries, in the next sections we will describe an interval-based representation of time series using interval boxes. In addition, we show how this representation can be used to efficiently support interval-focused similarity search using an existing index structure. The key benefit of our novel representation is that the query algorithm does not have to access the complete time series in a first filter step. Instead, the filter can work an relevant interval boxes which yield a lower and upper bound for the exact L_p distance.

6.3 Index Support for Interval-Focused Queries

The straightforward approach to calculate interval-focused distance values is to load all time series from the database into main memory and calculate the distance for the relevant stretches of time. This approach obviously transfers a lot of data that is not necessary for the computation of the result set of interval-focused queries. This is especially true if the relevant portions are relatively small compared to the length of the complete time series.

The basic idea of our approach is to represent each time series object of the database by sequences of intervals. These intervals can be efficiently managed by an index such as the RI-tree [KPS01a]. In addition, if we store the maximum and minimum amplitude of the time series within the intervals, these intervals can be used to compute upper and lower bounds of the true distance between different time series. If an interval-focused similarity query is launched specifying a set of relevant time frames $\mathcal{I}$, only the intervals of the database objects that intersect any $I \in \mathcal{I}$ need to be accessed in order to estimate the lower and upper bounding distance approximations. An overview of our approach is depicted in Figure 6.2. In a preprocessing step the available time series are approximated by intervals. The intervals are stored in an RI-tree. For the two relevant intervals, an intersection query is performed on the RI-tree. The resulting approximations can then be used to calculate a lower bounding and an upper bounding distance for the time series objects of the database. Thus, only a few time series objects have to

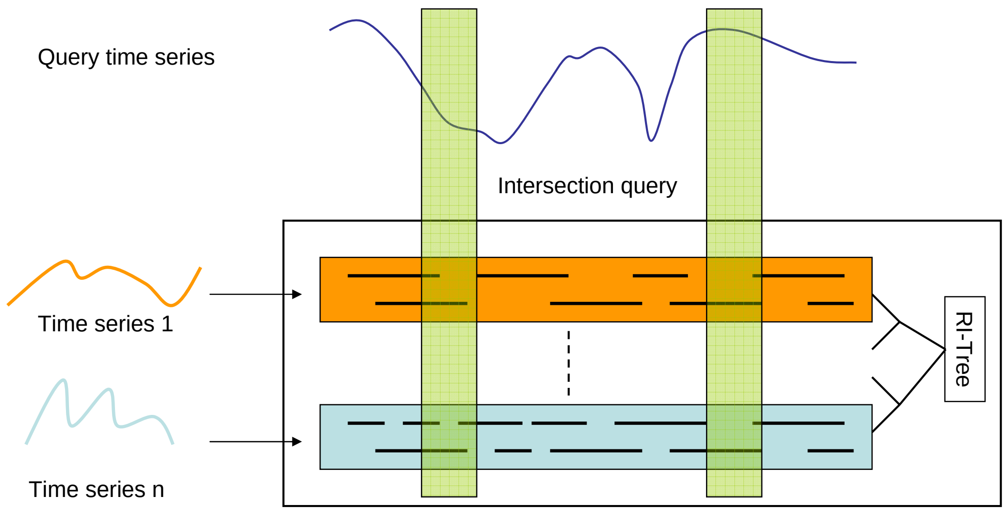


Figure 6.2: Overview of interval-focused similarity search.

be refined using the complete exact information.

6.3.1 Representing Time Series by Interval Boxes

As outlined above, we approximate each time series $X \in \mathcal{D}$ by a set of intervals. For each interval, we further store the maximal and minimal amplitude value of X within the interval. This results in a minimal-bounding box for X within the specified interval (see Figure 6.3) called interval box.

Definition 6.6 (Interval Box).
Let X be a time series and let (l_r, u_r) be a time interval. The interval box *r is given by the quadruple $r = (l_r, u_r, lv_r, uv_r)$, where lv_r and uv_r are the minimal and maximal amplitude values within I, i.e.*

$$lv_r = \min_{l_r \leq i \leq u_r} (x_i) \qquad uv_r = \max_{l_r \leq i \leq u_r} (x_i)$$

As we will outline later, we try to approximate a time series by a set of interval boxes. The set of interval boxes approximating X is denoted by $rep(X)$. Methods for generating interval boxes for a given time series are described in Section 6.3.3.

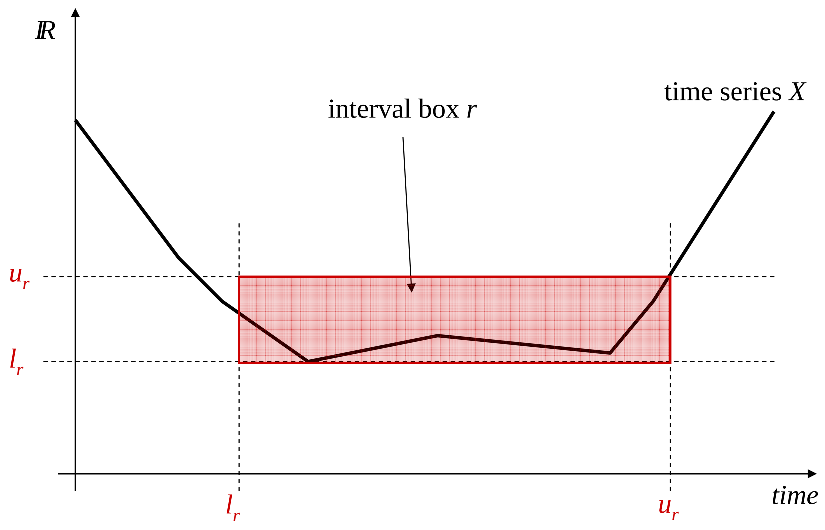


Figure 6.3: Interval box approximation of a time series.

6.3.2 Distance Estimation Using Interval Boxes

Let us assume, a time series is represented by a set of interval boxes. In this section we show how the exact interval-focused distance between a query object Q and any $X \in \mathcal{D}$ can be estimated by means of an upper and a lower bound using the information of $rep(X)$.

At each relevant time slot i, we can lower bound the i-th summand of the L_p-norm by the well-known $MINDIST$ function.

Definition 6.7 (MINDIST).
Let Q be a query time series and q_i the amplitude value of Q at time slot t_i. Let $X \in \mathcal{D}$ be a time series and let $r \in rep(X)$ be an interval box that overlaps t_i, i.e. $l_r \leq t_i \leq u_r$. Then the $MINDIST$ between q_i and $r = (l_r, u_r, lv_r, uv_r)$ is defined as

$$MINDIST(q_i, r) = \begin{cases} lv_r - q_i & \textit{if} \quad q_i \leq lv_r \\ q_i - uv_r & \textit{if} \quad q_i \geq uv_r \\ 0 & \textit{else.} \end{cases}$$

So, $MINDIST(q_i, r)$ lower bounds the exact value of the summand of the L_p distance at time slot t_i. This will be formally shown in the proof for Lemma 6.1.

In case no interval box $r \in rep(X)$ is available that overlaps time slot t_i, we can only lower bound the true distance between q_i and x_i by 0. If there are several interval boxes $r \in rep(X)$ with $l_r \leq i_t \leq u_r$, we determine the maximal value over all the corresponding MINDIST values. We will use the MINDIST value to define a lower bound for the interval-focused similarity. In order to get a tight lower bound (i.e. a value as high as possible), the maximal possible MINDIST value has to be used at every time slot.

Definition 6.8 (Lower Bound at a Single Time Slot).
Let Q and X be time series of length N and let X be represented by a collection of interval boxes $rep(X)$. Then $LB^i(Q,X)$ for time slot i is defined as

$$LB^i(Q,X) = \max\{0, \max_{\{r \mid r \in rep(X), l_r \leq i \leq u_r\}} (MINDIST(q_i, r))\}.$$

Now we extend the lower bound at each time slot i to an interval.

Definition 6.9 (Lower Bound for a Single Interval).
Let X and Q be time series and let $I = (l_I, u_I)$ be a time interval. Then the lower bound $LB^I(Q,X)$ is defined as

$$LB^I(Q,X) = \sqrt[p]{\sum_{i=l_I}^{u_I} (LB^i(Q,X))^p}.$$

Finally we can define a lower bound value for a set of non-overlapping relevant time intervals.

Definition 6.10 (Lower Bound for Interval-Focused Similarity).
Let X and Q be time series and let $\mathcal{I}$ be a set of relevant time intervals. Then the lower bound $LB^{\mathcal{I}}(Q,X)$ is defined as

$$LB^{\mathcal{I}}(Q,X) = \sqrt[p]{\sum_{I \in \mathcal{I}} (LB^I(Q,X))^p}.$$

Lemma 6.1 (Lower Bounding Property of $LB^{\mathcal{I}}$).
Let X and Q be time series and let $\mathcal{I}$ be a set of relevant time intervals. Then $LB^{\mathcal{I}}(Q,X)$ is a lower bound for $L_p^{\mathcal{I}}(Q,X)$, i.e.

$$LB^{\mathcal{I}}(Q,X) \leq L_p^{\mathcal{I}}(Q,X)$$

Proof. *Let X and Q be two time series. Let $I = (l_r, u_r)$ be a time interval. Let us assume an overlapping time interval box $r = (l_r, u_r, lv_r, uv_r)$ exists, i.e. $\forall x_i, l \leq i \leq u : lv \leq x_i \leq uv$.*
At first, we show $MINDIST(q_i, r) \leq |q_i - x_i|$:

1) $q_i \geq uv_r$: $MINDIST(q_i, r) = q_i - uv_r \leq q_i - x_i \leq |q_i - x_i|$ *(because $uv_r \geq x_i$)*

2) $q_i \leq lv_r$: $MINDIST(q_i, r) = lv_r - q_i \leq x_i - q_i \leq |q_i - x_i|$ *(because $lv_r \leq x_i$)*

3) $lv_r < q_i < uv_r$: $MINDIST(q_i, r) = 0 \leq |q_i - x_i|$.

So, $MINDIST(q_i, r) \leq |q_i - x_i|$. This holds for all $r \in rep(X)$. If no box is available, the summand for the corresponding time slot equals 0 (see Definition 6.8). Therefore $LB^i(Q,X) \leq |q_i - x_i|$. It follows

$$\sum_{i=l_j}^{u_j} (LB^i(Q,X))^p \leq \sum_{i=l_j}^{u_j} |q_i - x_i|^p$$

This equivalent to $(LB^I(Q,X))^p \leq (L_p^I(Q,X))^p$. We apply this observation to a sequence of intervals I:

$$\sum_{I \in \mathcal{I}} (LB^I(Q,X))^p \leq \sum_{I \in \mathcal{I}} (L_p^I(Q,X))^p$$

$$\Downarrow$$

$$\sqrt[p]{\sum_{I \in \mathcal{I}} (LB^I(Q,X))^p} \leq \sqrt[p]{\sum_{I \in \mathcal{I}} (L_p^I(Q,X))^p}$$

$$\Downarrow$$

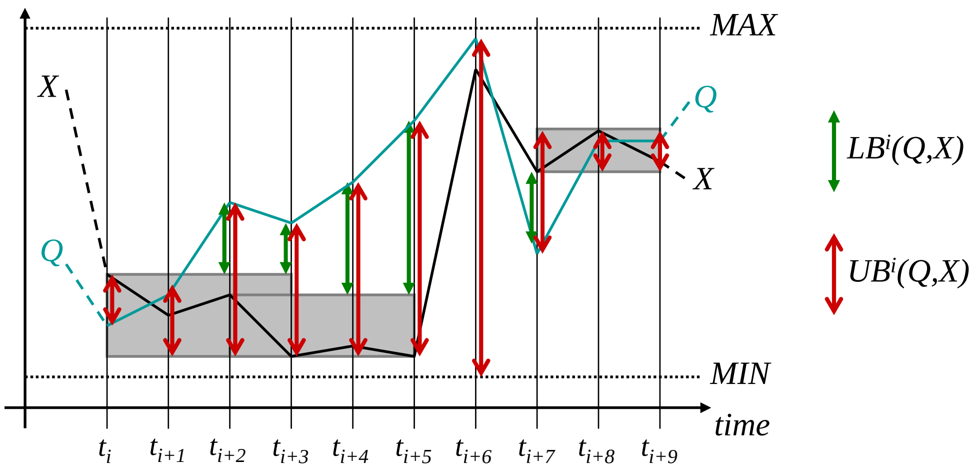


Figure 6.4: Lower and upper bounding the L_p-distance within the interval (t_i, t_{i+9}).

$$LB^{\mathcal{I}}(Q, X) \le L_p^{\mathcal{I}}(Q, X)$$

□

Analogously, an upper bounding distance estimation can be defined. At each relevant time slot i, we now need to use the $MAXDIST$ between q_i and any interval box $r \in rep(X)$ that overlaps i to define an upper bound of the i-th summand of $L_p(Q, X)$. The $MAXDIST$ function is defined as follows:

Definition 6.11 (MAXDIST).
Let Q be a query time series and q_i the amplitude value of Q at time slot t_i. Let $X \in \mathcal{D}$ be a time series and let $r \in rep(X)$ be an interval box that overlaps t_i, i.e. $l_r \le t_i \le u_r$. Then the $MAXDIST$ value between q_i and $r = (l_r, u_r, lv_r, uv_r)$ is defined as

$$MAXDIST(q_i, r) = \max\{|q_i - lv_r|, |q_i - uv_r|\}$$

If there is an interval box $r \in rep(X)$ that overlaps time slot t_i, we can upper bound the true distance between q_i and x_i using the MAXDIST value. If there are several interval boxes $r \in rep(X)$ with $l_r \le t_i \le u_r$, we compute the minimum over all the MAXDIST values to derive the best possible approximation to the actual distance. If no overlapping box is available, we can only estimate the upper bound for a certain time slot by a value $u := max\{|q_i - MAX|, |q_i - MIN|\}$.

Definition 6.12 (Upper Bound at a Single Time Slot).
Let Q and X be time series of length N and let X be represented by a collection of interval boxes $rep(X)$. Then $UB^i(Q,X)$ for time slot i is defined as

$$UB^i(Q,X) = min\{u, \min_{\{r \mid r \in rep(X), l_r \leq i \leq u_r\}} (MAXDIST(q_i, r))\}$$

Now we extend the definition of the upper bound at a single time slot i to an interval.

Definition 6.13 (Upper Bound for a Single Interval).
Let X and Q be time series and let $I = (l_I, u_I)$ be a time interval. Then the upper bound $UB^I(Q,X)$ is defined as

$$UB^I(Q,X) = \sqrt[p]{\sum_{i=l_I}^{u_I} (UB^i(Q,X))^p}.$$

Finally we can define an upper bound value for a set of non-overlapping relevant time intervals.

Definition 6.14 (Upper Bound for Interval-Focused Similarity).
Let X and Q be time series and let $\mathcal{I}$ be a set of relevant time intervals. Then the upper bound $UB^{\mathcal{I}}(Q,X)$ is given by

$$UB^{\mathcal{I}}(Q,X) = \sqrt[p]{\sum_{I \in \mathcal{I}} (UB^I(Q,X))^p}.$$

Lemma 6.2 (Upper Bounding Property of $UB^{\mathcal{I}}$).
Let X and Q be time series and let $\mathcal{I}$ be a set of relevant time intervals. Then $UB^{\mathcal{I}}(Q,X)$ is an upper bound for $L_p^{\mathcal{I}}(Q,X)$, i.e.

$$UB^{\mathcal{I}}(Q,X) \geq L_p^{\mathcal{I}}(Q,X)$$

Proof. *This can be shown analogously to the proof for Lemma 6.1.* □

An example for the upper and lower bounding distance estimation is depicted in Figure 6.4. At time slot t_{i+6} no interval box representation is available for X. So, the lower bound can only be estimated by $LB^{t_{i+6}}(Q,X) = 0$. The value for the upper bound can only be estimated as $UB^{t_{i+6}}(Q,X) = \max\{|q_{t_{i+6}} - MAX|,\ |q_{t_{i+6}} - MIN|\}$. In contrast to that, at time t_{i+1} the interval box $r = (t_i, t_{i+3}, lv_r, uv_r) \in rep(X)$ approximates the time series X. So we can estimate $LB^{t_{i+1}}(Q,X) = MINDIST(q_{t_{i+1}}, r) = 0$ and $UB^{t_{i+1}}(Q,X) = MAXDIST(q_{t_{i+1}}, r) = |q_{t_{i+1}} - lv_r|$.

6.3.3 Generating Approximations

In this section, we will show how to generate suitable interval boxes for a time series. When generating the interval boxes we need to take two conflicting considerations into account. On one hand, the number of boxes covering the time series should be low in order to avoid a high overhead effort during the filter step. The more interval boxes are present, the more calculations have to be performed and the more boxes have to be loaded into main memory. So, the computational cost and the IO cost of the filter step is influenced by the number of interval box approximations to be considered at query time. This suggests to construct wide boxes with long intervals.

On the other hand, wide boxes will usually worsen the approximation quality since the boxes conservatively approximate the time series. As a consequence, the performance may decrease due to a reduced pruning power of the filter step. This leads to higher refinement cost in the next step. A refinement actually corresponds to accessing the complete time series, something we actually wanted to prevent. This observation suggests to construct boxes with low approximation error in order to achieve higher values for the lower bounding filter distance $LB^{\mathcal{I}}$ and lower values for the upper bounding filter distance $UB^{\mathcal{I}}$.

Following these considerations, the sections of the time series having a flat

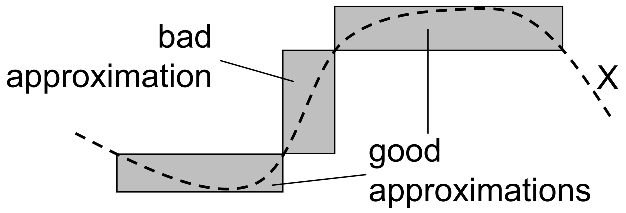


Figure 6.5: Interval box approximations.

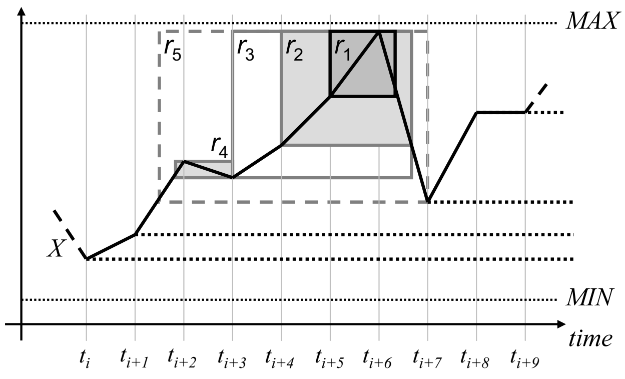


Figure 6.6: Generation of covering boxes.

curvature can be better approximated by interval boxes than parts featuring a high ascending or descending curve. An example for this observation is depicted in Figure 6.5. The basic idea of our approach is to optimize the box-covering locally. We first identify those parts of the time series which can be well approximated, i.e. subsequences covering the local maximums or minimums of a time series. Then, we generate interval boxes that optimally cover the local minimums and maximums of a time series according to a quality criterion given below. Afterwards, we approximate each remaining part of the time series which are not covered yet by one single box.

A high approximation quality of the interval box approximations of a time series is responsible for a good pruning power during the filter step (see Section 6.4). A large lower bounding distance estimation allows to prune a lot of true drops without the need to refine them. A small upper bounding distance estimation allows to identify some of the true hits without any refinement. For this reason we evaluate the approximation quality of an interval box by considering the expected value of the lower and upper bounding distance between any query object and the approximated part of the database object. Actually, for two intervals I and J we compare their corresponding values $(LB^I)^p$ and $(LB^J)^p$ instead of LB^I and LB^J, as it is more efficient to compute $(LB^I)^p$ and $(LB^J)^p$. This does not change the process of deciding for the better suited interval, as obviously $(LB^I)^p \geq (LB^J)^p \Rightarrow LB^I \geq LB^J$ holds.

In order to calculate the expected value of $(LB^I)^p$, we have to make an assumption concerning the distribution of amplitude values of potential query time series. The first assumption is, that the occurring amplitude values are uniformly distributed between MIN and MAX. The second assumption is that the amplitude values for different time slots are not correlated. Both assumptions will probably not hold for a single query time series of a real-world dataset. However, as we are interested to generate an approximation which is well suited for a broad range of query time series, we are more interested in the average query time series. Hence, our assumptions may very well hold for such an average time series.

Note furthermore that it is possible to either optimize the expected value of the lower bound, or the upper bound, as they are correlated. In the following we will show how to optimize the lower bound.

Lemma 6.3 (Expected Value of $(LB^I)^p$).
Let X and Q be two time series. Let $I = (l_j, u_j)$ be a time interval and let $r = (l_j, u_j, lv_j, uv_j)$ be the corresponding time interval box, i.e. $\forall x_i, l_j \leq i \leq u_j : lv_j \leq x_i \leq uv_j$. Let furthermore the amplitude values of Q be statistically independent from each other and let the amplitude values of Q be equally distributed between MIN and MAX. Then the expected value $E((LB^{(I)}(Q,X))^p)$ is given by

$$E((LB^I(Q,X))^p) = (u_j - l_j + 1) \cdot \frac{(l_j - MIN)^{p+1} + (MAX - u_j)^{p+1}}{(MAX - MIN)(p+1)}$$

Proof. *Let X, Q, and r be defined as above. In order to calculate $E(LB^I)^p$,* *at first we replace LB^I according to its definition.*

$$E((LB^I(Q,X))^p) = E(\sum_{i=l_j}^{u_j}(LB^i(Q,X))^p)$$

As the expected value of a sum of random variables can be calculated as the sum of the expected values of the single random variables, we get

$$E(\sum_{i=l_j}^{u_j}(LB^i(Q,X))^p) = \sum_{i=l_j}^{u_j} E((LB^i(Q,X))^p)$$

According to the definition of the expected value we get

$$E((LB^i(Q,X))^p) \quad = \quad \int_{MIN}^{MAX}(MINDIST(q_i,r))^p \cdot f_i(q_i)\mathrm{d}q_i$$

Under the assumption that the values q_i of Q are equally distributed between MIN and MAX, the probability density function is constant for all q_i, i.e.

$$f_i(q_i) = \frac{1}{MAX - MIN}, \forall q_i \in [MIN, MAX].$$

Depending on the position of q_i relative to the box r, we have to distinguish 3 cases for the value of MINDIST and so we split the integral into a sum of three integrals.

$$\begin{aligned}
E((LB^i(Q,X))^p) &= \frac{\left(\int_{MIN}^{l_j}(l_j-q_i)^p\mathrm{d}q_i+\int_{l_j}^{u_j}0^p\mathrm{d}q_i+\int_{u_j}^{MAX}(q_i-u_j)^p\mathrm{d}q_i\right)}{MAX-MIN} \\
&= \frac{\left[-\frac{(l_j-q)^{p+1}}{p+1}\right]_{MIN}^{l_j}+0+\left[\frac{(q-u_j)^{p+1}}{p+1}\right]_{u_j}^{MAX}}{MAX-MIN} \\
&= \frac{0+\frac{(l_j-MIN)^{p+1}}{p+1}+\frac{(MAX-u_j)^{p+1}}{p+1}-0}{MAX-MIN} \\
&= \frac{(l_j-MIN)^{p+1}+(MAX-u_j)^{p+1}}{(MAX-MIN)(p+1)}
\end{aligned}$$

As the value for $E((LB^i(Q,X))^p)$ is independent of i, we can calculate the result of $\sum_{i=l_j}^{u_j} E((LB^i(Q,X))^p)$ as

$$\sum_{i=l_j}^{u_j} E((LB^i(Q,X))^p) = (u_j - l_j + 1) \cdot \frac{(l_j - MIN)^{p+1} + (MAX - u_j)^{p+1}}{(MAX - MIN)(p+1)}$$

□

This lemma actually describes the tradeoff between the width of an interval and its height. Now, we can use the expectation of the distance estimations in order to decide for an interval box whether the box setting is more promising than alternative box settings. The higher the expected lower bounding distance w.r.t. an interval box approximation, the higher is its approximation quality.

As already mentioned, flat parts, like the local maximums or minimums, of a time series are very adequate for our interval box approximation. We start with the approximation of the local maximums of a time series by searching for each local maximum iteratively in top-down direction. For each local maximum we take all reasonable conservative coverings into account as shown in the example depicted in Figure 6.6. In this example we start with considering the box r_1. Following the top-down direction, the next box to be considered is r_2, followed by r_3. In this example, r_2 is the best alternative.

When the algorithm continues, box r_4 is created for a different local maximum. The next box that is generated is box r_5. Now the algorithm simply calculates the sum of the expected contribution to the lower distance bound of boxes r_2 and r_4 and compares this value to the expected value for r_5. In case r_5 is better suited, the boxes r_2 and r_4 are discarded, as they are completely covered by r_5 and we try to minimize the amount of stored boxes. This procedure will be applied to all local maximums, so that finally all local maximums are covered by an interval box.

Afterwards the coverings of the local minimums are generated in the same way, except this time we start at the local minimums and search the corresponding interval box candidates in an upward direction. After generating all local maximum and minimum coverings, we remove those box candidates which are completely covered by another interval box candidate in order to reduce redundant approximations. Note that due to the two generation steps, it is possible that the boxes overlap. However, this does not interfere with the definitions of the lower and upper bounds. In this case, the better suited approximation for a given query will be used.

Finally in a post-processing step, the remaining gaps between two adjacent but disjunctive interval boxes, i.e. the parts of the time series which are not covered so far by any interval box, are simply approximated by an additional minimal bounding box.

6.4 Interval-Focused Query Processing

In order to compute the upper and lower bounding distance approximations between a query object Q and a database object $X \in \mathcal{D}$ efficiently, we have to determine those interval boxes that intersect the relevant intervals $I \in \mathcal{I}$. For the efficient support of interval intersection queries, we organize the intervals of the interval boxes in an adoption of the relational interval tree (RI-tree) [KPS01a]. An interval intersection query takes a query interval $I \in \mathcal{I}$ and retrieves all intervals in the RI-tree that intersect with I. Details on the processing of intersection queries using RI-Trees can be found in [KPS01a]. In order to determine all interval boxes that intersect with the query intervals we need such an intersection query for all $I \in \mathcal{I}$. This way, we determine for each database object $X \in \mathcal{D}$ those interval boxes $r \in rep(X)$ that intersect with any of the query intervals $I \in \mathcal{I}$ in order to compute $LB^{\mathcal{I}}(Q, X)$ and $UB^{\mathcal{I}}(Q, X)$.

Based on the distance approximations $LB^{\mathcal{I}}$ and $LB^{\mathcal{I}}$ introduced above, we can apply the paradigm of filter-refinement query processing to efficiently answer interval-focused distance range and kNN queries. In case of an interval-focused distance range query, we can use both, the upper and the lower bound in the filter step. Each object $X \in \mathcal{D}$ with $LB^{\mathcal{I}}(Q, X) > \varepsilon$ can be identified as true drop because $L_p^{\mathcal{I}}(Q, X) \geq LB^{\mathcal{I}}(Q, X) > \varepsilon$, i.e. $X \notin IQ_{\varepsilon}^{range}(Q, \mathcal{I})$. On the other hand, each object $X \in \mathcal{D}$ with $UB^{\mathcal{I}}(Q, X) \leq \varepsilon$ can be identified as true hit since $L_p^{\mathcal{I}}(Q, X) \leq UB^{\mathcal{I}}(Q, X) \leq \varepsilon$, i.e. $X \in IQ_{\varepsilon}^{range}(Q, \mathcal{I})$. The pseudocode for an interval-focused distance range query is depicted in Figure 6.7.

```
IQRQ($Q$, $\varepsilon$, $\mathcal{I}$, $\mathcal{D}$)
  /*Intersect the relevant intervals with the database intervals*/
  $L_{Cand}$ = RI-Tree.intersect($\mathcal{I}$);
  /*$L_{Cand}$ contains a set of intersecting intervals for each database object*/
  $result = \emptyset$;
  while $L_{Cand} \neq \emptyset$ do
    $Boxes$ := $L_{Cand}$.removeFirstElement();
    $X$ := time series represented by $Boxes$
    /*Calculate upper and lower bounds, using $Boxes$*/
    $LB^{\mathcal{I}}$ := calculateLowerBound($Q$,$Boxes$);
    if $LB^{\mathcal{I}} > \varepsilon$ then
        /*prune current time series $X$, continue with next while-loop*/
    else
        $UB^{\mathcal{I}}$ := calculateUpperBound($Q$,$Boxes$);
        if $UB^{\mathcal{I}} \leq \varepsilon$ then
            result.add($X$)
        else
            /*refine time series and decide based on exact distances*/
  return $result$;
```

FIGURE 6.7: Pseudocode for the interval-focused range query.

In case of an interval-focused kNN query, we can only use the lower bound for the filter step. We apply the approach presented in [SHP98] which is optimal with respect to the number of candidates that need to be refined. The idea of the approach in [SHP98] is to dynamically update the filter criterion, whenever the exact distance of the k^{th} nearest neighbor retrieved so far is calculated. This distance can be used as the new filter distance. If an object which has not yet been refined has a lower bounding distance larger than the filter distance, it can not be an element of the resulting kNN set.

6.5 Evaluation

In this section, we examine the efficiency of our proposed indexing technique for interval-focused queries.

6.5.1 Datasets and Methods

As outlined above, our approach is especially suited for large time series with only a relatively small relevant fraction of time. We used two real-world datasets of the audio collection described in Chapter 3 for our comparison. They were created using two different feature extraction methods on a song database. This resulted in two different dataset which will be referred to as "DS1" and "DS2" in the following. Both consist of 4,800 time series of length 10,000. As the feature extraction methods differ for the datasets, the typical elements of both datasets differ as well.

All experiments were performed on a workstation featuring a 1.8 GHz Opteron CPU and 8GB RAM. We used a disk with a transfer rate of 60 MB/s, a seek time of 3 ms, a latency delay of 2 ms. The node capacity of the RI-tree was set to 8 KByte. For each experiment we randomly selected 100 sample queries and averaged the observed results over all queries. For our efficiency evaluation we counted several events. For index-based approaches like our approach and competitors that use index structures as well, we counted the visited directory nodes of the RI-tree and the data leafs that had to be transferred to the main memory. Using the technical specifications of our hard disk we calculated the required IO time to answer the query. Whenever we compared our approach to the sequential scan approach, we calculated the time it would have taken to transfer the stored time series into the main memory, as in this case no index structure and no lower or upper bounds are used.

Note we did not report the actual time for the calculation of the distance value as we observed that the differences for the competing approaches were too small to be measured reliable. This is due to the fact that roughly the same number of calculations has to be performed for all techniques. The straightforward approach using a sequential scan simply sums up the distance values for all considered time slots. This approach can not rely on lower or upper bound, so actually all relevant time slots have to be considered. On the other hand, the index-based approaches can exploit lower and upper bounds. Based on the knowledge of bounding boxes for complete stretches of time slots, the contribution to the distance value of a certain stretch in time can efficiently be estimated with only a few calculations. Of course, extra time has to be spent for the traversal of the index structure. In summary, we could not measure any meaningful differences in terms of computation time. The significant differences for different approaches could be observed in terms of the IO time.

We executed queries for varying focus sizes, i.e. for varying relevant portions of the complete time series objects. The focus size is specified as a percentage of the length of the complete time series. This value is the sum of all relevant intervals, as we randomly created several non-overlapping relevant intervals to define the relevant time slots.

6.5.2 Experimental Results

Interval Box Generation

We first evaluate our method for generating interval box representations in comparison to two more straightforward solutions. The first competing approach determines intervals for the interval box generation randomly. We set the number of randomly created non-overlapping intervals to the number our approach yields for an average time series. This technique will be labeled with "RANDOM" in the following.

The second competing approach splits the time series in a number of time stretches of equal length. For each of these constant-sized intervals, a minimal bounding interval box is generated as described above. We label this approach with "EQUAL" throughout this section. The EQUAL method corresponds to the box generation of the PAA approximation technique [YF00].

Our method will be referred to as “OTPIMAL” in the following.

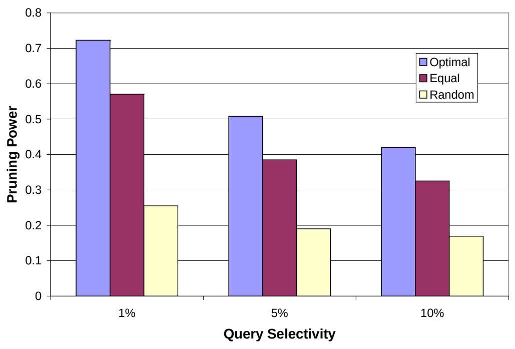
At first we measured the pruning power of the three techniques. The pruning power is measured as the relative amount of time series that have to be refined after their interval boxes were used to calculate upper and lower bounds. In case the upper or lower bounds allow for a sure inclusion or a sure exclusion of the current time series element for the result set, the pruning power is increased. So, the maximal value for the pruning is 1 and the minimal and worst value is 0. A pruning power of 0 means that all time series objects have to be refined.

The pruning power results for DS1 are displayed in Figure 6.8 for two different focus size values (2% and 5%). Furthermore we varied the value for the ε value of the interval-focused ε-range query. Depending on the choice of ε, the result set of the query has a different cardinality. The number of elements of the result set relative to the size of the complete database can be considered as the query selectivity. In order to allow for a better comparison between the two datasets, we searched for appropriate ε values that yielded a certain predefined selectivity.

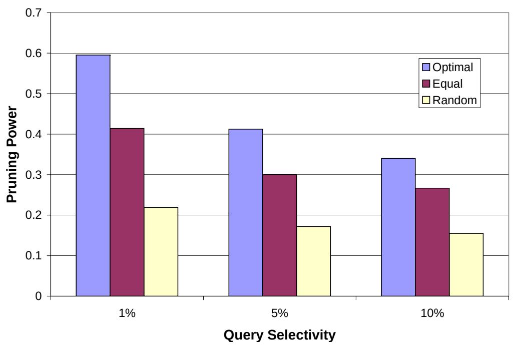
The results show two interesting properties. First, the pruning power varied for different focus sizes. For all compared approaches, the pruning power is higher for a smaller focus size. Second, the pruning power varied for different query selectivity values. Higher query selectivity values correspond to larger ε values and larger focus sizes lead to larger distance values. These larger distance and ε values lead to more time series that have to be refined, as only a smaller portion of all time series has a very small distance (and so, a very small filter distance based on the interval boxes) to the query object. So, the percentage of time series that can be excluded or included solely based on the filter step decreases with increasing focus length and with increasing ε values.

However, our proposed box generation method clearly outperforms the competing approaches for a broad range of parameter settings.

Examining the pruning power is not sufficient, it is rather necessary to consider the real IO cost as well. For example it is possible that the intervals



(a) Focus Size 2%



(b) Focus Size 5 %

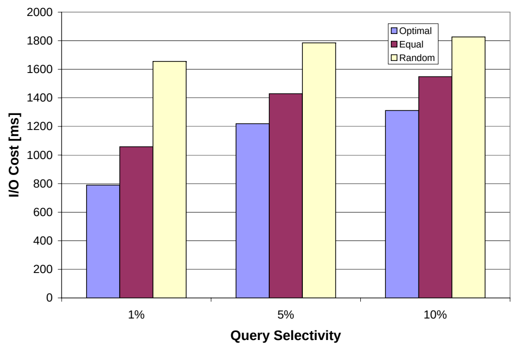
FIGURE 6.8: Evaluation of different interval box generation methods (pruning power).

of one of the other approaches can be more efficiently organized in the RI-tree and so the access cost for this approach could be lower. So, in Figure 6.9 the IO cost as explained above are depicted for DS1. The same experiment was repeated for the second dataset (see Figure 6.10). The same observation as for the pruning power experiments can be made with respect to different focus size and varying query selectivity.

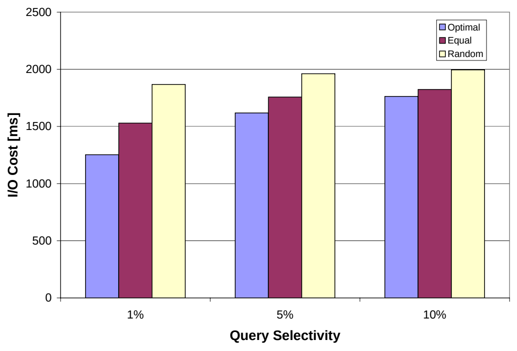
Query Selectivity

In a next set of experiments we evaluated our complete filter-refinement architecture. Our approach will again be labeled with “OPTIMAL”. We compared our results to the sequential scan. This is the straightforward approach where the time series are sequentially loaded into the main memory and the relevant portions of the time series are used to calculate the appropriate distance values. These corresponding results are labeled with “SEQ. SCAN”. The third technique included in our experiments is the approach proposed in [RKBL05] and will be labeled with “BIT LEVEL”. We chose the second competitor since it is the only approach that does not need to scan the entire time series information for answering interval-focused queries but also proposes a filter-refinement architecture based on compressed data representation.

In Figure 6.11 the results for DS1 and two different focus size value is

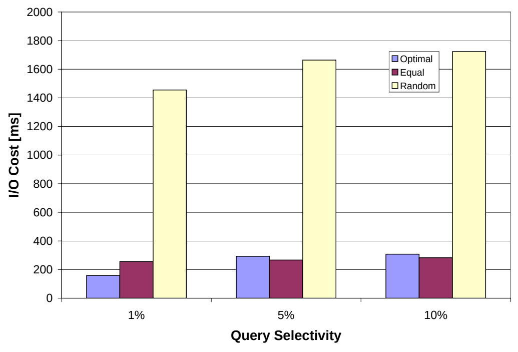


(a) Focus Size 2%

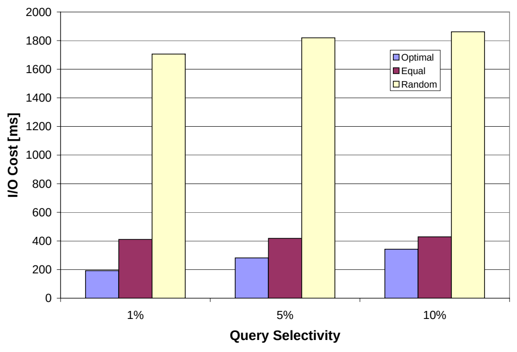


(b) Focus Size 5%

FIGURE 6.9: Total I/O cost of different interval box generation methods (DS1).

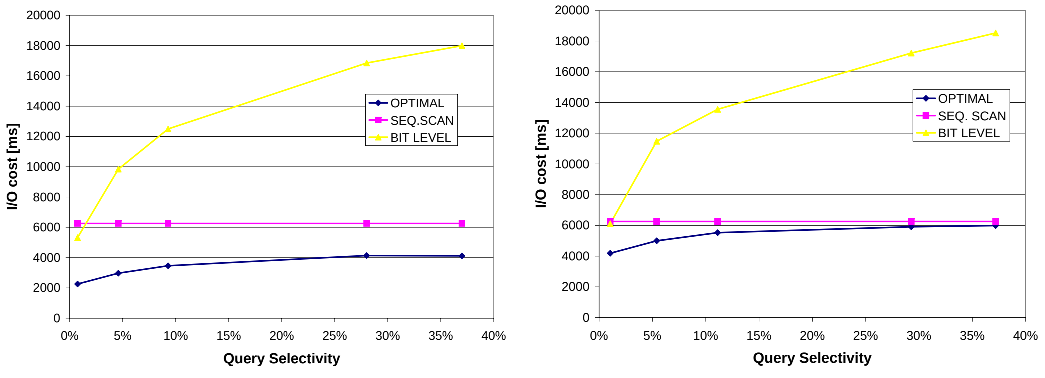


(a) Focus Size 2%



(b) Focus Size 5%

FIGURE 6.10: Total I/O cost of different interval box generation methods (DS2).



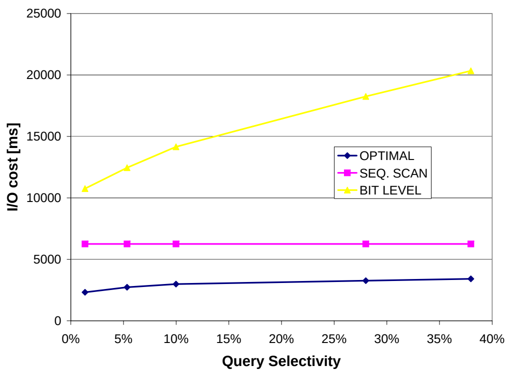
(a) Focus Size 2% (b) Focus Size 4%

Figure 6.11: Performance with respect to the selectivity of the query for two different values for the focus size (DS1).

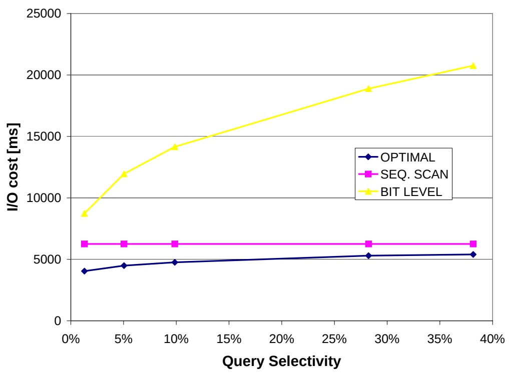
given. In Figure 6.12 the results for DS2 are presented. Our approach clearly outperforms both competitors for all settings of the query selectivity. Furthermore our approach scales significantly better for increasing query selectivity compared to the bit level approach. The results for the sequential scan are obviously independent of focus size ore query selectivity, as for all settings the same amount of time series has to be transferred to the main memory. Again, we can observe that our approach is more suitable for smaller values for the focus size on both datasets. The absolute values for the IO cost differ on both datasets, as our approach generates the interval boxes depending on the actual time series values which are different for the two datasets.

Focus Size

As we already noticed the importance of the focus size, we performed a third set of experiments examining the influence of the focus size. The results in terms of the total IO cost are depicted in Figure 6.13 for DS1 and in Figure 6.14 for DS2. On both datasets and for both considered query selectivity values our approach is more efficient than the sequential scan up to a focus size of approximately 6%. This value may be different for other datasets and other query selectivity values but confirms our general assumption. Interval-



(a) Focus Size 2%



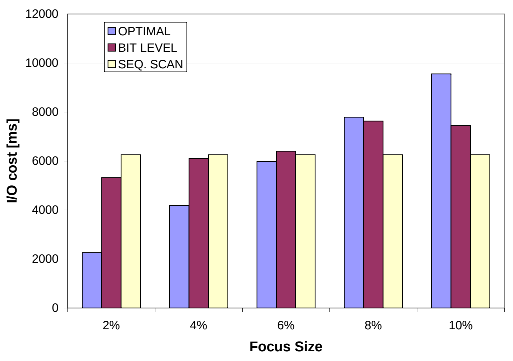
(b) Focus Size 4%

FIGURE 6.12: Performance with respect to the selectivity of the query for two different values for the focus size (DS2).

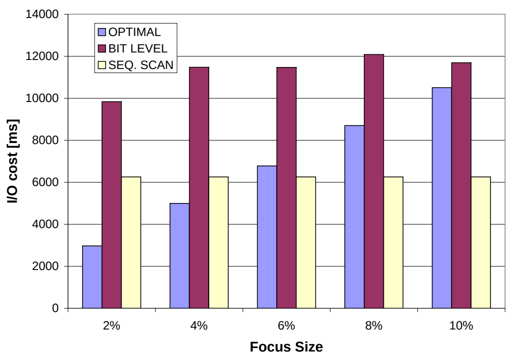
focused queries are especially useful for very long time series objects where only a relatively small portion is of interest. This is due to the additional IO cost for the random access during the traversal of the RI-tree index and to the random access for the refinement of single time series. If the focus size is increased, more random accesses will occur in the RI-tree intersection query. If the selectivity value is increased, more refinements will be necessary as explained above. However, in many real-world a focus size smaller than 6% is a reasonable query size. Think of querying for a certain week in year which corresponds to a focus size of less than 2%. Depending on the actual technical system and on the characteristics of the dataset, even the focus on a certain month may be more efficiently answered using our interval-focused query approach.

6.6 Conclusions

In this chapter, we introduced the concept of interval-focused similarity queries in time series databases,an important generalization of comparing entire time series. We introduced a new efficient representation of time series based on intervals and showed how this representation can be used to efficiently support these new query type implementing a filter-refinement ap-

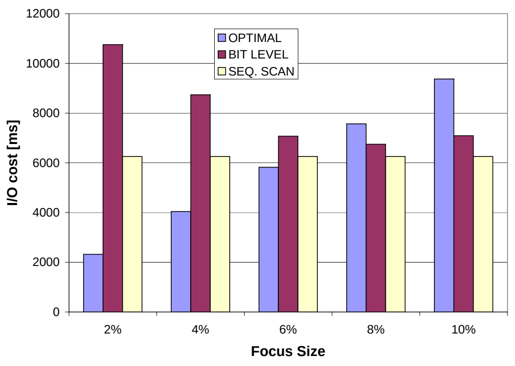


(a) Query Selectivity 1%

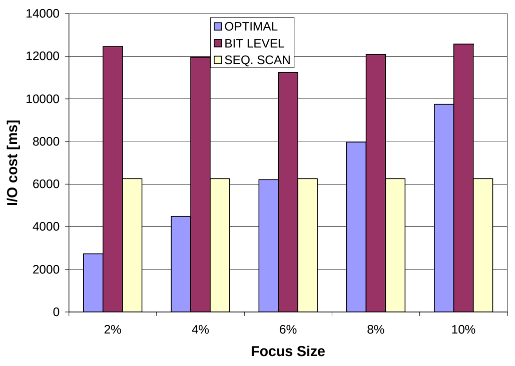


(b) Query Selectivity 5%

FIGURE 6.13: Performance with respect to the size of the query focus for two different query selectivity values(DS1).



(a) Query Selectivity 1%



(b) Query Selectivity 5%

FIGURE 6.14: Performance with respect to the size of the query focus for two different query selectivity values(DS2).

proach. An important property of our approach is that the relevant intervals can be specified at query time and do not need to be known in advance. Furthermore, we presented a method for the generation of the interval-based representation. In our experimental evaluation we showed the superiority of our proposed method for answering interval-focused similarity queries in comparison to existing approaches as long as the portion of relevant intervals is not too large compared to the complete time series. The exact break-even value between the cost for the sequential scan compared to our index-based method depends on the used hardware. Faster disk transfer rates decrease the cost for the sequential scan approach. Lower cost for random access on the other hand, decreases the cost for our index-based approach. However, the focus size up to which our method outperforms competing approaches is in a range suitable for a lot of real-world problems.

Chapter 7

Similarity Search on Uncertain Time Series

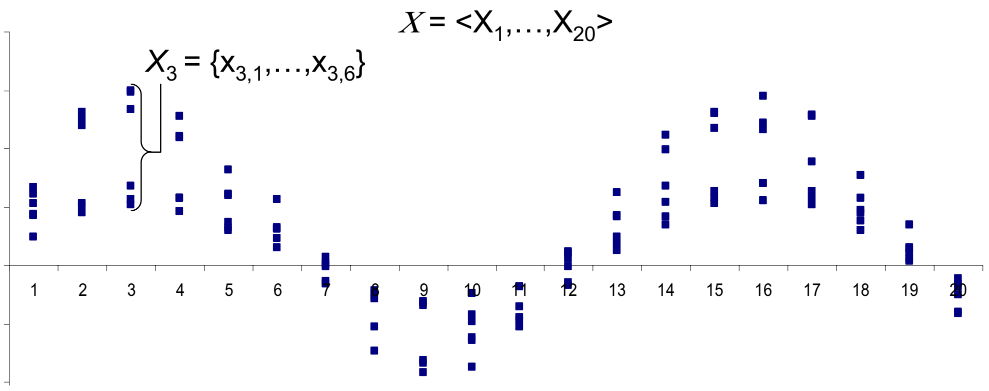
In the previous chapters we have introduced several new similarity measures for time series. In all these chapters we have considered regular time series where exactly one amplitude value is stored for each time slot. As we have outlined in Chapter 2, a lot of previous work has focused on such regular time series. However, fewer techniques are available to support queries on uncertain time series. Uncertainty is important in emerging applications dealing for example with moving objects or object identification as well as sensor network monitoring. In all these applications, the observed values at each time slot of a time series exhibit various degrees of uncertainty. Due to the uncertainty of the data objects, similarity queries are probabilistic rather than exact: we can only assign to each database object a probability that it fulfills the query predicate. Traditional approaches only consider uncertainty as a probability density function and are furthermore not designed for time series. For situations where several measurements for each time slot are available, no similarity measure and no efficient query processing has been proposed so far. Applications where the analysis of time series has to cope with uncertainty include:

Traffic sensors located at different roads usually measure the amount of traffic rather inaccurately due to technical reasons. In most cases, for each sensor a model that specifies the possible error of the measurement is provided. Finally the observed values do not consist of a single observation but are given by a set of possible observations.

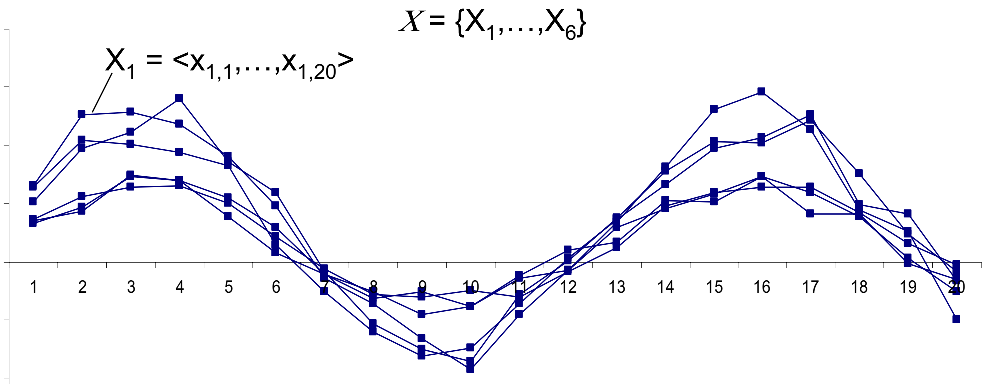
Environmental observations such as temperature or the concentration of particulate matter are monitored during each day of the year at different sites. Finding sites with similar daily monitored measurements is important in order to analyze relationships between the measured parameters and climatic phenomena. Assume there are measurements over 5 years available for a certain parameter. If a researcher wants to analyze the considered parameter over the course of a year, each site has associated five values with it for each time slot during a day. These five measurements at each time slot represent all possible values of this site at the particular time slot. In addition, each of these five observed values at time slot i is correlated with one of the five observed values at time slot $(i + 1)$ because they have been observed in the same year.

In database systems managing moving objects, the position of these objects is usually updated periodically. Between every two time slots of update, the position of any object o is uncertain within a small spatial range of the latest observed location of o. So, a number of possibilities exist for the exact position of an object.

In all these applications, methods for similarity search have to consider that the observed values are uncertain and may additionally be correlated. In the literature, two principal possibilities to model uncertainty are discussed. First, uncertainty can be expressed by a probabilistic density function (PDF), e.g. a uniform or a given normal distribution. Such a PDF specifies the probability a certain value is observed. If an uncertain time series T is represented by PDFs, at each time slot i a PDF is used to model the probability distribution for T having a certain value at i. The second way of modeling uncertainty, which we will focus on in this chapter, is a set of alternative values along with corresponding probability values as proposed in [LLRS97]. This kind of representation is motivated by the fact that often multiple sensors are used to study the same phenomenon. The concurring results from these



(a) Uncorrelated uncertain time series.



(b) Correlated uncertain time series.

Figure 7.1: Uncertain time series.

sensors are fused to provide an estimate of the observed object. There are a lot of application areas where multiple sensor sources are merged (sensor fusion). The result of such a fusion is a number of possible alternatives (samples), each of which has an associated probability. If an uncertain time series T is represented by alternative samples, for each time slot i a set of sample values is given, representing possible values of T at i. Another advantage of the sample-based uncertainty representation is that, whereas in the first approach the used PDFs must be fixed and known in advance, sampling can model any distribution without prior knowledge. In addition, we can easily generate a sample-based representation based on a given PDF. The reverse way however, is not always possible, as not every possible set of points can be mathematically described by a PDF. So, the sample-based representation is the more flexible and more general way of representing uncertainty. For a given set of observations, it may be even wrong to model these observations by a PDF, as the PDF does not necessarily model the true underlying process the observations where created with. On the other hand, the sample-based representation means sticking to the facts.

The above mentioned applications correspond to two kinds of sample-based uncertain time series that are illustrated in Figure 7.1. In the traffic sensor example and the moving objects example, the sample values of different time slots are uncorrelated, i.e. there is no relationship between a given sample observation at time slot i and another sample observation at time slot $(i+1)$ (see Figure 7.1(a)). Think of the traffic sensor example. At time i the positions of several cars may have been recorded. At the next time slot, a completely different set of cars may have come in view. While the general position of the new set of cars may depend on the cars at the earlier time slot (depending on how fast the old cars move), the exact position of a car at $i + 1$ does not depend on the exact position of a single specific car at i. In contrast to that scenario, think of the environmental example. Here, each observed sample at time slot i is correlated to an observation at time slot $(i + 1)$ and vice versa (see Figure 7.1(b)). Here it is possible to establish a relationship between two single samples at successive time slots, because they were recorded at the same site. Both types of uncertainty require different solutions in order to support probabilistic similarity queries. The uncorrelated representation is the more general representation. It can easily be created based on an correlated representation by removing information. Hence, in this thesis we focus on uncorrelated uncertain time series.

To the best of our knowledge, this is the first work, to deal with probabilistic similarity queries over uncertain time series. In particular, the contributions of our work are as follows:

- We formalize the problem of probabilistic queries on uncertain time series, focusing on two types of probabilistic range queries in Section 7.3.
- We propose a novel compact and approximate representation of uncer-

tain time series.

- We show how upper and lower bounding distance approximations for the Euclidean distance and the DTW distance can be derived based on these representations in Section 7.5.
- We illustrate how these distance approximations can be used to design a multi-step query algorithm for efficient probabilistic similarity queries on uncertain time series in Section 7.5.
- In an experimental evaluation we demonstrate the performance boost of our approach compared to competing solutions in Section 7.6.

7.1 Related Work

7.1.1 Similarity Search on Time Series

As time series have become an increasingly prevalent type of data, a lot of work on similarity search in time series databases has been published (cf. Chapter 2). However, all these approaches deal with regular time series and do not consider any uncertainty.

In the case of uncertain time series, the approaches proposed in Chapter 2 cannot be applied. The main reason is that the uncertainty in each value of a time series leads to an uncertain distance between time series. Existing approaches for supporting similarity search on time series cannot deal with that fact because they implicitly assume that the distance between time series returns a single value.

7.1.2 Similarity Search on Uncertain Vector Objects

The fact that time series can be considered as a point in n-dimensional space suggests that uncertain time series could be treated as n-dimensional uncertain vectors. Several approaches for indexing uncertain vector objects have been proposed, mainly differing in the type of the uncertainty and in the type of similarity query supported [CKP03, CXP$^+$04, TCX$^+$05, BPS06b, BPS06a, CKP07, LS07]. These approaches deal with an uncertainty model for spatially uncertain objects and propose queries which are specified by intervals in the query space. In this setting, a query retrieves uncertain objects that are located within the query interval with a certain likelihood. Other approaches deal with statistical modeling of data in sensor networks [FGB02, DGM$^+$04, DGM05]. However, all mentioned approaches use continuous probability density functions for the description of the spatial uncertainty. Thus, these approaches rely on the assumption that the uncertainty can be modeled by a specific PDF, usually a Gaussian distribution. As discussed above, it is more general to model uncertainty by means of sample observations rather than PDFs. This way, any uncertainty distribution can be modeled. Approaches for similarity search on uncertain spatial objects using the sampling model are proposed in [KKPR06, KKM07]. The proposed approaches allow to approximate uncertain objects represented by arbitrarily structured PDFs. The sampled positions in space can efficiently be indexed using traditional spatial access methods. This reduces the computational complexity of complicated query types.

In general, all these approaches are not applicable for uncertain time series. First, the proposed techniques are not designed for high-dimensional spaces. As mentioned above, time series are usually very high-dimensional feature vectors. To the best of our knowledge, there has no dimensionality reduction technique for uncertain data been proposed so far. The main problem of indexing uncertain objects in high-dimensional spaces is the low query selectivity due to the curse of dimensionality. Second, only correlated uncertain time series can be treated as n-dimensional uncertain vectors. Similarity search of uncorrelated uncertain time series is not covered by the discussed approaches. Third, none of the above-mentioned approaches is able to support DTW. Last but not least, all these approaches are designed to reduce IO cost, although the challenge when handling uncertain time series is to reduce CPU cost, as we will explain in the following sections.

7.2 Special Notations

As we introduce a lot of new notations in this chapter, we give a brief overview of frequently used abbreviations and notations.

7.3 Probabilistic Similarity Queries for Uncertain Time Series

- X: regular time series
- $\mathcal{X}$: uncertain time series
- X_t: set of sample points at time slot t
- $\mathcal{X}_a$: approximative representation of $\mathcal{X}$
- X_a: sequence of n d-dimensional boxes
- $I_{t,j}$: d-dimensional minimal bounding box (mbr) for a given number of sample points of X_t
- $s_{\mathcal{X}}$: sample size of $\mathcal{X}$ (depending on the context also "s")
- d: dimensionality of samples points
- n: time series length
- $TS_{\mathcal{X}}$: set of all possible regular time series obtained by combining the sample points of $\mathcal{X}$
- $TS_{\mathcal{X}_a}$: set of all possible combination of different mbrs of $\mathcal{X}_a$
- L_p: Minkowski distance for parameter p
- DTW_p: DTW distance based on Minkowski norm with parameter p
- $dist_{L_p}(\mathcal{X}, \mathcal{Y})$: collection of all L_p distances between $TS_{\mathcal{X}}$ and $TS_{\mathcal{Y}}$

- $dist_{DTW_p}(\mathcal{X}, \mathcal{Y})$: collection of all DTW_p distances between $TS_{\mathcal{X}}$ and $TS_{\mathcal{Y}}$
- $dist_p(\mathcal{X}, \mathcal{Y})$: $dist_{L_p}(\mathcal{X}, \mathcal{Y})$ or $dist_{DTW_p}(\mathcal{X}, \mathcal{Y})$
- $\Pr(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$: probability that the distances between two uncertain time series $\mathcal{X}$ and $\mathcal{Y}$ are below threshold ε
- τ: probability value
- $RQ_{\varepsilon,\tau}(\mathcal{Q}, \mathcal{D})$: probabilistic bounded range query on dataset $\mathcal{D}$
- $RQ_{\varepsilon,\text{rank}}(\mathcal{Q}, \mathcal{D})$: probabilistic ranked range query on dataset $\mathcal{D}$

7.3.1 Uncertain Time Series

At first we formalize the notion of uncorrelated uncertain time series and introduce two important query types for uncertain time series. Usually, time series are sequences of (certain) d-dimensional points. Uncertain time series are sequences of points having an uncertain position in the d-dimensional vector space. This uncertainty is represented by a set of observations at each time slot.

Definition 7.1 (Uncertain Time Series).
An uncertain time series $\mathcal{X}$ *of length* n *consists of a sequence* $\langle X_1, \ldots, X_n \rangle$. *Each element* X_t *contains a set of* s *d-dimensional points (sample observations), i.e.* $X_t = \{x_{t,1}, \ldots, x_{t,s}\}$ *with* $x_{t,i} \in \mathbb{R}^d$. *We call* s *the* sample size *of* $\mathcal{X}$. *The distribution of the points in* X_t *reflects the uncertainty of* $\mathcal{X}$ *at time slot* t.

We use the term *regular time series* for traditional, non-uncertain time series consisting of only a single d-dimensional point at each time slot. To improve the presentation, we assume 1-dimensional uncertain time series in the following. However, the extension of the concepts presented in this thesis to the general d-dimensional case is straightforward.

7.3.2 Uncertain Distances

In order to measure the similarity of uncertain time series, a distance measure for such uncertain time series is required. For regular time series, the Minkowski distance (L_p-norm) is commonly used to measure the distance between time series. Due to the uncertainty of the time series, the distance between two time series becomes uncertain as well. Instead of computing a single distance value such as the L_p-norm of the corresponding sequences, the distance between uncertain time series consists of multiple distance values reflecting the distribution of all possible distance values between the samples of the corresponding uncertain time series. This intuition is formalized in the following definition.

Definition 7.2 (Uncertain L_p-Distance).
For a one-dimensional uncertain time series $\mathcal{X}$ of length n, let $s_\mathcal{X}$ be the sample size of $\mathcal{X}$ and $TS_\mathcal{X}$ be the set of all possible regular time series that can be derived from the combination of different sample points of $\mathcal{X}$ by taking one sample from each time slot, i.e.

$$\begin{aligned} TS_\mathcal{X} \;=\{\;& \langle x_{1,1}, x_{2,1}, \ldots, x_{n,1} \rangle, \\ & \langle x_{1,2}, x_{2,1}, \ldots, x_{n,1} \rangle, \\ & \langle x_{1,1}, x_{2,2}, \ldots, x_{n,1} \rangle, \\ & \ldots \\ & \langle x_{1,s_\mathcal{X}}, x_{2,s_\mathcal{X}}, \ldots, x_{n,s_\mathcal{X}} \rangle \quad \}. \end{aligned}$$

*The L_p-*distance *between two uncertain time series $\mathcal{X}$ and $\mathcal{Y}$, denoted by $dist_{L_p}$, is a collection containing the L_p distances of all possible combinations from $TS_\mathcal{X}$ and $TS_\mathcal{Y}$, i.e.*

$$dist_{L_p}(\mathcal{X}, \mathcal{Y}) = \{L_p(x, y) \mid x \in TS_\mathcal{X}, y \in TS_\mathcal{Y}\}.$$

An example for the set $TS_\mathcal{X}$ is depicted in Figure 7.2. The set $dist_{L_p}(\mathcal{X}, \mathcal{Y})$ can be calculated by computing the L_p distances for all possible combinations of elements from $TS_\mathcal{X}$ and $TS_\mathcal{Y}$. In this example, $dist_{L_p}(\mathcal{X}, \mathcal{Y})$ contains 16 distance values.

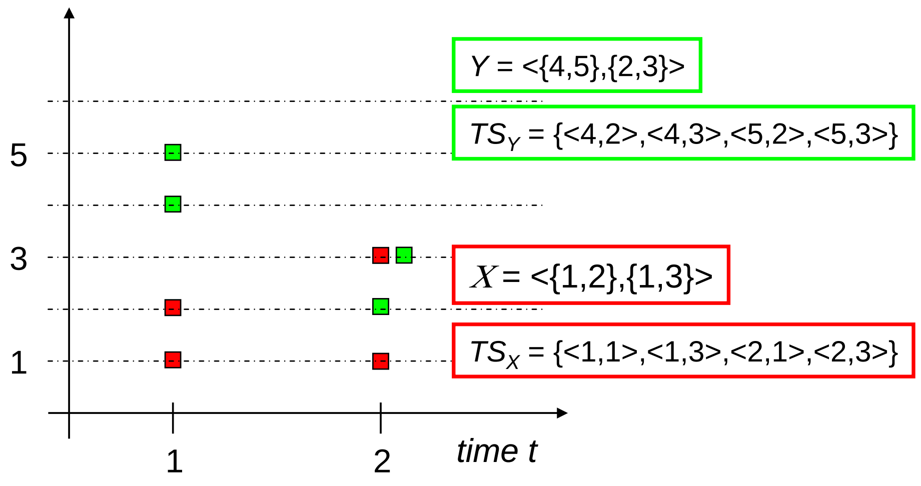


FIGURE 7.2: Computation of the uncertain distance set.

The L_p-norm is the most prominent similarity measure for (regular) time series, in particular for $p \in 1, 2$. However, often dynamic time warping (DTW) is used. Compared to the Euclidean distance, the DTW allows small distortions in time. Analogously to Definition 7.2, we formalize the DTW-distance for uncertain time series in the following.

Definition 7.3 (Uncertain DTW-Distance).
Let DTW_p be the DTW distance for regular time series based on an arbitrary L_p-norm. The DTW-distance between two uncertain time series *$\mathcal{X}$ and $\mathcal{Y}$, $dist_{DTW_p}$, is a collection containing the DTW distances of all possible combinations from $TS_{\mathcal{X}}$ and $TS_{\mathcal{Y}}$, i.e.*

$$dist_{DTW_p}(\mathcal{X}, \mathcal{Y}) = \{DTW_p(x, y) \,|\, x \in TS_{\mathcal{X}}, y \in TS_{\mathcal{Y}}\}.$$

In the following, we assume that $dist_p$ is either the L_p-distance or the DTW-distance between uncertain time series, i.e. $dist_p \in \{dist_{L_p}, dist_{DTW_p}\}$. Based on the distance function $dist_p$ we define two query types for uncertain time series.

Lemma 7.1 (Cardinality of Distance Set).
Given two uncertain time series $\mathcal{X}$ and $\mathcal{Y}$ of length n and sample sizes $s_{\mathcal{X}}$ and $s_{\mathcal{Y}}$, respectively. Then the distance set between $\mathcal{X}$ and $\mathcal{Y}$ contains $s_{\mathcal{X}}^n \cdot s_{\mathcal{Y}}^n$ elements (sample distance observations), i.e. $|dist_p(\mathcal{X}, \mathcal{Y})| = s_{\mathcal{X}}^n \cdot s_{\mathcal{Y}}^n$.

Proof. *Obviously, $|dist_p| = |TS_{\mathcal{X}}| \cdot |TS_{\mathcal{Y}}|$ because it contains the distances between all possible combinations of elements in $TS_{\mathcal{X}}$ and $TS_{\mathcal{Y}}$. Since $\mathcal{X}$ is of length n and has a sample size of $s_{\mathcal{X}}$, $|TS_{\mathcal{X}}| = s_{\mathcal{X}}^n$. Analogously, $|TS_{\mathcal{Y}}| = s_{\mathcal{Y}}^n$. Thus, $|dist_p| = s_{\mathcal{X}}^n \cdot s_{\mathcal{Y}}^n$.* □

7.3.3 Probabilistic Similarity Queries

Based on the distance between two uncertain time series, we can define two important types of probabilistic similarity queries for uncertain time series extending the concept of distance range queries on regular time series. These queries are probabilistic because we cannot for sure report whether the distance of two objects is lower than a given threshold ε. We can only determine the probability that the distance between two time series is lower than ε.

The probability $\Pr(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$ that the distance between two uncertain time series $\mathcal{X}$ and $\mathcal{Y}$ is below a given threshold ε is the fraction of distance observations in $dist_p$ that are below or equal ε, formally:

$$\begin{aligned}\Pr(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon) &= \frac{|\{d \in dist_p(\mathcal{X}, \mathcal{Y}) | d \leq \varepsilon\}|}{|dist_p(\mathcal{X}, \mathcal{Y})|} \\ &= \frac{|\{d \in dist_p(\mathcal{X}, \mathcal{Y}) | d \leq \varepsilon\}|}{s_{\mathcal{X}}^n \cdot s_{\mathcal{Y}}^n}\end{aligned}$$

The above statement is based on the assumption that the sample points at each time slot of an uncertain time series $\mathcal{X}$ reflect a set of $s_{\mathcal{X}}$ observed values, each occurring with a probability of $\frac{1}{s_{\mathcal{X}}}$. Here, we do not regard whether the samples are based on an underlying distribution or are reported from different sensors. We assume that the samples are the only available information concerning the uncertain time series. As a consequence, given two uncertain time series $\mathcal{X}$ and $\mathcal{Y}$ of length n, each distance $d \in dist_p(\mathcal{X}, \mathcal{Y})$ has the same probability $\frac{1}{s_{\mathcal{X}}^n \cdot s_{\mathcal{Y}}^n}$, because $s_{\mathcal{X}}^n \cdot s_{\mathcal{Y}}^n$ possible distances arise from $\mathcal{X}$ and $\mathcal{Y}$.

The first query type called *probabilistic bounded range query* returns all time series that have a distance less than a given ε to the query with a probability of at least τ. Both ε and τ are specified by the user at query time.

Definition 7.4 (Probabilistic Bounded Range Query).
Let $\mathcal{D}$ be a database of uncertain time series, $\varepsilon \in \mathbb{R}^+$, and $\tau \in [0,1]$. For an uncertain query time series $\mathcal{Q}$, the Probabilistic Bounded Range Query *(PBRQ) returns the set $PQ_\varepsilon^{range}(\mathcal{Q},\tau)$ such that*

$$\forall \mathcal{X} \in PQ_\varepsilon^{range}(\mathcal{Q},\tau) : \Pr(dist_p(\mathcal{Q},\mathcal{X}) \le \varepsilon) \ge \tau.$$

The second query type called *probabilistic ranked range query* returns a ranking of the uncertain database time series with respect to the probability $\Pr(dist_p(\mathcal{Q},\mathcal{X}) \le \varepsilon)$ that the corresponding time series has a distance less than ε to the query time series.

Definition 7.5 (Probabilistic Ranked Range Query).
Let $\mathcal{D}$ be a database of uncertain time series and $\varepsilon \in \mathbb{R}^+$. For an uncertain query time series $\mathcal{Q}$, the Probabilistic Ranked Range Query *(PRRQ) returns an ordered list $PQ_\varepsilon^{rank}(\mathcal{Q},m) = (\mathcal{X}_1,\ldots,\mathcal{X}_m)$ where*

$$\forall i = 1,\ldots,m-1 : \Pr(dist_p(\mathcal{Q},\mathcal{X}_i) \le \varepsilon) \ge \Pr(dist_p(\mathcal{Q},\mathcal{X}_{i+1}) \le \varepsilon)$$

Furthermore we define a function *getNext* on the list $PQ_\varepsilon^{rank}(\mathcal{Q},\tau)$ that returns the next element of the ranking. Instead of calculating the complete ranking for all database objects only the first few required elements of the list are computed.

Very often, the time series in a database are relatively long. It is also not uncommon that the uncertain time series are recorded with a high sample rate. So, the naive solution for both query types is CPU-bound because for all $\mathcal{X} \in \mathcal{D}$ we have to compute all distance observations in $dist_p(\mathcal{Q},\mathcal{X})$ in order to determine $\Pr(dist_p(\mathcal{Q},\mathcal{X}) \le \varepsilon)$. Due to Lemma 7.1, this means that a naive solution requires to compute for each $\mathcal{X} \in \mathcal{D}$ exactly $|dist_p(\mathcal{Q},\mathcal{X})| = s_\mathcal{Q}^n \cdot s_\mathcal{X}^n$ distances. For large values of n, $s_\mathcal{Q}$, and $s_\mathcal{X}$, this is much more costly than sequentially scanning the disk to access all $\mathcal{X} \in \mathcal{D}$. For example, assuming standard hardware parameters like a seek time of 3 ms, a latency delay of 2 ms, a transfer rate of 80MB/s, and 50 ns for one Euclidean distance computation (of two 1D points), the CPU cost of the naive solution exceed the IO cost of the naive solution by a factor of approximately 10^{200} ($n = 100$, $s_{\mathcal{Q}} = s_{\mathcal{X}} = 10$, $|\mathcal{D}| = 10,000$) when using the Euclidean distance for $dist_p$. Using the DTW for $dist_p$, this factor is even higher.

A first idea for runtime reduction is that we only need to determine the number of distance observations $d \in dist_p(\mathcal{Q}, \mathcal{X})$ with $d \leq \varepsilon$ because the complete number of distance possible observations is known according to Lemma 7.1 and equals $s_{\mathcal{Q}}^n \cdot s_{\mathcal{X}}^n$. We can further improve the runtime by calculating lower and upper bounds for the probability that further reduce the number of distance computations. For that purpose, we have to calculate an upper and a lower bound for the number of distance observations $d \in dist_p(\mathcal{Q}, \mathcal{X})$ with $d \leq \varepsilon$.

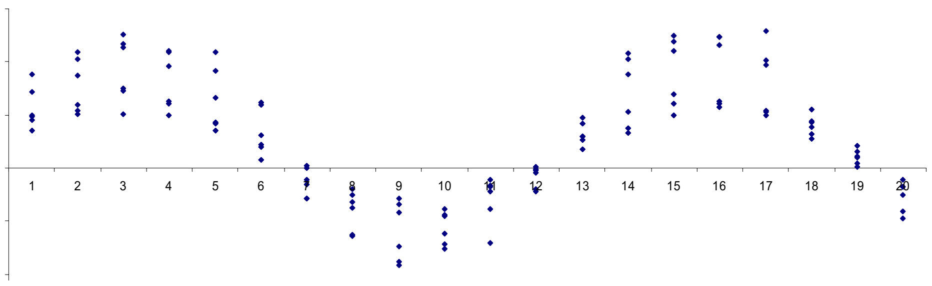
In the following, we first introduce an approximative representation of uncertain time series. We will then illustrate how these approximations can be used to upper and lower bound the distance observations $d \in dist_p(\mathcal{Q}\mathcal{X})$ which can subsequently be used to identify lower and upper bounds for $\Pr(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon)$.

7.4 Approximative Representation for Uncertain Time Series

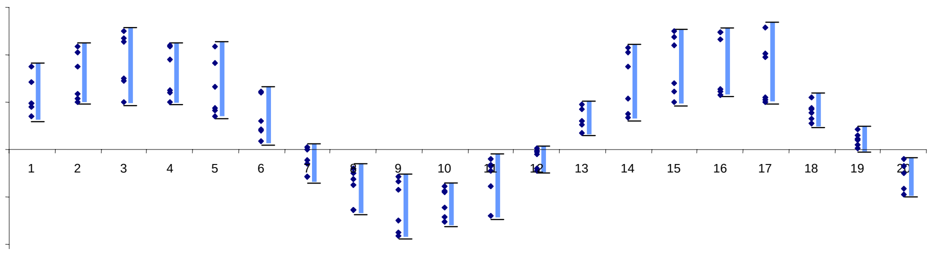
7.4.1 Efficient Representation of Uncertain Time Series

We construct the approximative representation of an uncertain time series $\mathcal{X}$ by aggregating the observations $x_{i,j} \in X_i$ at each time slot i into groups and use these groups to calculate the distance between uncertain time series. Obviously, this reduces the sample rate and thus, the overall number of possible distance combinations. The groups are represented by minimum bounding intervals or minimum bounding hyper-rectangles in the d-dimensional case.

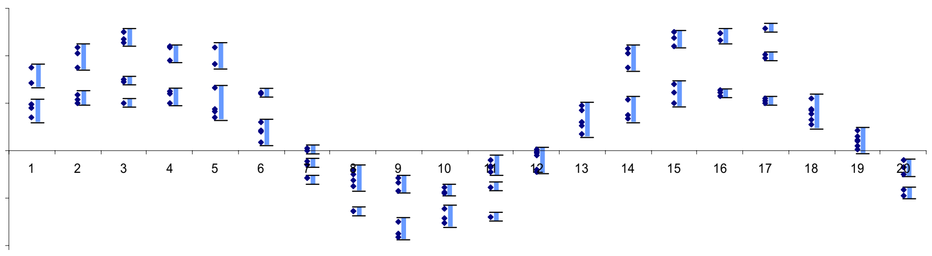
Definition 7.6 (Approximative Representation).



(a) Exact representation.



(b) 1^{st}-level approximation.



(c) 2^{nd}-level approximation.

Figure 7.3: Different approximation levels of uncertain time series.

The approximative representation $\mathcal{X}_a$ *of an uncertain time series* $\mathcal{X}$ *of length* n *consists of a sequence* $\langle\{I_{1,1}, \ldots, I_{1,m_1}\}, \ldots, \{I_{n,1}, \ldots, I_{n,m_n}\}\rangle$ *of interval sets. Each interval* $I_{i,j} = [l_{i,j}, u_{i,j}]$ *minimally covers a certain number* $|I_{i,j}|$ *of sample points of* X_i*, i.e.* $l_{i,j}$ *and* $u_{i,j}$ *are sample points of* X_i*, at time slot* i*.*

We use two approximation levels. The first level describes all sample points at time slot i by one minimal bounding interval (see Figure 7.3(b)), i.e. $m_i = 1$ for all time slots i and $\mathcal{X}_a = \langle I_{1,1}, \ldots, I_{n,1}\rangle$. For the second-level approximations, the samples at time slot i are grouped into k clusters by applying the algorithm k-means [Mac67] on all $x_{i,j} \in X_i$ (cf. Figure 7.3(c)). In this case, $m_i = k$ for all time slots i and $\mathcal{X}_a = \langle\{I_{1,1}, \dots, I_{1,k}\}, \dots, \{I_{n,1}, \dots, I_{n,k}\}\rangle$. In fact, the exact representation of $\mathcal{X}$ can also be considered as a sequence of interval sets where $I_{i,j} = [x_{i,j}, x_{i,j}]$ and $m_i = s_{\mathcal{X}}$ for each time slot. Note that for a given state during the query algorithms presented in the next sections, the levels of approximation can differ for different time slots. That is, the approximated times series become partially refined.

7.4.2 Approximating Distances

Using approximative representations $\mathcal{X}_a$ and $\mathcal{Y}_a$ of two uncertain time series $\mathcal{X}$ and $\mathcal{Y}$, we are able to calculate lower and upper bounds for $\Pr(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$.

Definition 7.7 (Approximated Regular Time Series).
Let X_a be the approximative representation of an uncertain time series $\mathcal{X}$. A sequence of exactly one approximation for each time slot is called an approximated regular time series X_a. Note that an approximated regular time series can contain exact sample points as well, as long as only a single element (be it an interval or a sample point) is specified for each time slot.

Two examples for approximated regular time series are depicted in Figure 7.4. In this example, each approximated regular time series is depicted by a path of boldly printed intervals.

Analogously to Definition 7.2, let $TS_{\mathcal{X}_a}$ be the set of all possible approximated regular time series that can be derived by combining different intervals of $\mathcal{X}_a$ by selecting one interval from each time slot, i.e.

$$\begin{aligned} TS_{\mathcal{X}_a} = \{ \quad & \langle I_{1,1}, I_{2,1}, \dots, I_{n,1}\rangle, \\ & \langle I_{1,2}, I_{2,1}, \dots, I_{n,1}\rangle, \\ & \langle I_{1,1}, I_{2,2}, \dots, I_{n,1}\rangle, \\ & \dots \\ & \langle I_{1,l_1}, \dots, I_{n,l_n}\rangle \quad \}. \end{aligned}$$

Definition 7.8 (Bounded Distances for Approximated Regular Time Series).

Let $X_a \in TS_{\mathcal{X}_a}$ and let $[l_{x_i}, u_{x_i}]$ be the interval of X_a at time slot i. The distance

$$L_{L_p}(X_a, Y_a) = \sqrt[p]{\sum_{i=1}^{n} (\max\{0, \max\{l_{x_i}, l_{y_i}\} - \min\{u_{x_i}, u_{y_i}\}\})^p}$$

is the smallest L_p-distance between all intervals of $X_a \in TS_{\mathcal{X}_a}$ and $Y_a \in TS_{\mathcal{Y}_a}$. The distance

$$U_{L_p}(X_a, Y_a) = \sqrt[p]{\sum_{i=1}^{n} (\max\{u_{x_i} - l_{y_i}, u_{y_i} - l_{x_i}\})^p}$$

is the largest L_p-distance between all intervals of $X_a \in TS_{\mathcal{X}_a}$ and $Y_a \in TS_{\mathcal{Y}_a}$.

For the DTW distance, L_{DTW_p} and U_{DTW_p} can be defined accordingly. In the following, we use the terms L_{dist} and U_{dist} as a generalization for both kinds of distances.

The concepts presented above are visualized in Figure 7.4 and in Figure 7.5. In Figure 7.4, two uncertain time series $\mathcal{X}$ and $\mathcal{Q}$ are depicted. Both contain a set of intervals at each time slot approximating the corresponding sample observations. One element $X_a \in TS_{\mathcal{X}_a}$ includes all boldly marked intervals of $\mathcal{X}$ and can be considered as one of many possible paths or approximated regular time series. The other path, $Q_a \in TS_{\mathcal{Q}_a}$, consists of all boldly marked intervals of $\mathcal{Q}$. The distance approximations between these two paths X_a and Q_a at each time slot i are depicted in Figure 7.5. Aggregating these distance values by means of the distance function $dist_p$, we obtain an interval of distances bounded by L_{dist} and U_{dist} (see Figure 7.6).

7.4.3 Approximating Probabilities

L_{dist} and U_{dist} allow us to define a lower and an upper bound for each element of $dist_p(\mathcal{X}, \mathcal{Y})$. Similarly to Definition 7.2, $LB_p(\mathcal{X}_a, \mathcal{Y}_a)$ is a collection containing the lower bounds L_{dist} of all possible combinations from $TS_{\mathcal{X}_a}$ and $TS_{\mathcal{Y}_a}$. Each element of $TS_{\mathcal{X}_a}$ represents a certain number of regular time

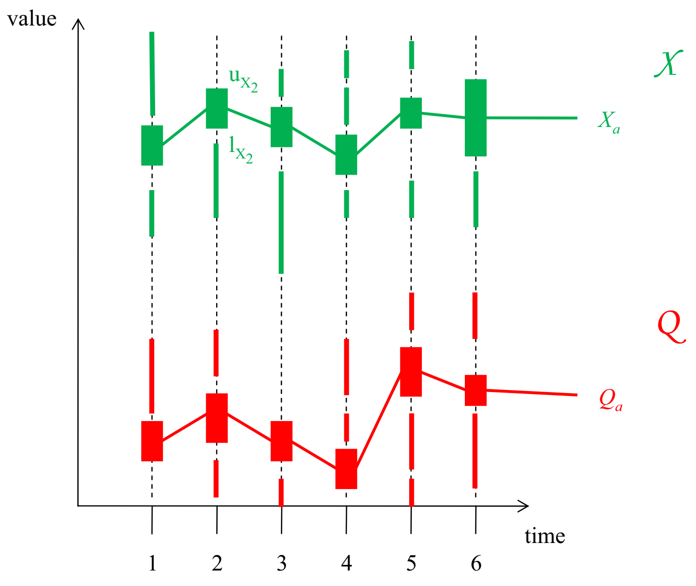


Figure 7.4: Approximated regular time series.

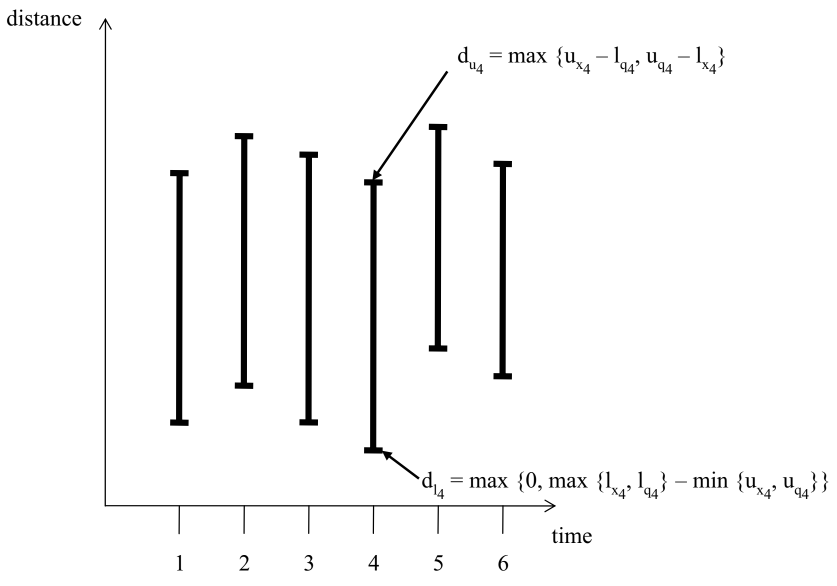


Figure 7.5: Calculating distance bounds for each time slot based on approximated regular time series.

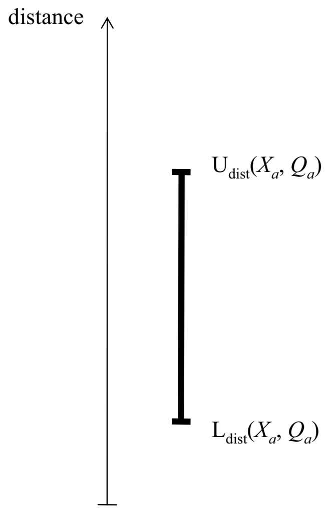


FIGURE 7.6: Final distance-bounding interval after the aggregation of the distance information for all time slots.

series that can be derived from the combination of different sample points by taking one sample point of each interval. So the number of regular time series represented by a given $X_a \in TS_{\mathcal{X}_a}$ is $|X_a| := \prod_{i=1}^{n} |I_i|$. Thus, when collecting L_{dist}-distances, for each possible combination $(X_a, Y_a) \in TS_{\mathcal{X}_a} \times TS_{\mathcal{Y}_a}$ the corresponding distance value $L_{dist}(X_a, Y_a)$ has to be stored $|X_a| \cdot |Y_a|$ times. As a consequence, we can lower bound each distance observation in $dist_p(\mathcal{X}, \mathcal{Y})$.

Definition 7.9 (Lower-Bounded Distance Set).
The set $LB_p(\mathcal{X}_a, \mathcal{Y}_a)$ contains a lower-bounded distance entry for each exact entry in $dist_p(\mathcal{X}, \mathcal{Y})$.

$$LB_p(\mathcal{X}_a, \mathcal{Y}_a) = \{(L_{dist}(X_a, Y_a))^{|X_a| \cdot |Y_a|} | X_a \in TS_{\mathcal{X}_a}, Y_a \in TS_{\mathcal{Y}_a}\}.$$

Analogously, we can upper bound each distance observation in $dist_p(\mathcal{X}, \mathcal{Y})$.

Definition 7.10 (Upper-Bounded Distance Set).
The set $UB_p(\mathcal{X}_a, \mathcal{Y}_a)$ contains an upper-bounded distance entry for each exact entry in $dist_p(\mathcal{X}, \mathcal{Y})$

$$UB_p(\mathcal{X}_a, \mathcal{Y}_a) = \{(U_{dist}(X_a, Y_a))^{|X_a| \cdot |Y_a|} | X_a \in TS_{\mathcal{X}_a}, Y_a \in TS_{\mathcal{Y}_a}\}.$$

Lemma 7.2 (Bounds Based on Approximated Regular Time Series).
Let $X_a = \langle I_1^x, \ldots, I_n^x \rangle \in TS_{\mathcal{X}_a}$ and $\mathcal{Y}_a = \langle I_1^y, \ldots, I_n^y \rangle \in TS_{\mathcal{Y}_a}$ be approximated regular time series. For all $x = \langle x_1, \ldots, x_n \rangle$, $x_i \in I_i^x$ and for all $y = \langle y_1, \ldots, y_n \rangle$, $y_i \in I_i^y$, the following inequalities hold:

$$L_{dist}(\mathcal{X}_a, \mathcal{Y}_a) \leq dist(x, y), \text{ where } dist \in \{L_p, DTW_p\}.$$

$$U_{dist}(\mathcal{X}_a, \mathcal{Y}_a) \geq dist(x, y), \text{ where } dist \in \{L_p, DTW_p\}.$$

Proof. *The lemma follows directly from the definition of L_{dist} and U_{dist}.* □

Finally we can define lower and upper bounds for the probability that two uncertain time series have a distance smaller than ε.

Definition 7.11 (Upper-Bounded Probability).
A lower bound for the probability $\Pr(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$ can be defined as

$$\Pr_{LB}(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon) = \frac{|\{d \in UB_p(\mathcal{X}_a, \mathcal{Y}_a) | d \leq \varepsilon\}|}{s_{\mathcal{X}}^n \cdot s_{\mathcal{Y}}^n}$$

Definition 7.12.
An upper bound for the probability $\Pr(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$ can be defined as

$$\Pr_{UB}(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon) = \frac{|\{d \in LB_p(\mathcal{X}_a, \mathcal{Y}_a) | d \leq \varepsilon\}|}{s_{\mathcal{X}}^n \cdot s_{\mathcal{Y}}^n}$$

Lemma 7.3 (Upper and Lower Bounding Property).
For any uncertain time series $\mathcal{X}$ and $\mathcal{Y}$, the following inequations hold:

$$(1) \quad \Pr_{LB}(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon) \leq \Pr(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$$

$$(2) \quad \Pr_{UB}(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon) \geq \Pr(dist_p(\mathcal{X}, \mathcal{Y}) \leq \varepsilon)$$

Proof.
(1) We have to show that $|\{d \in UB_p(\mathcal{X}_a, \mathcal{Y}_a) | d \leq \varepsilon\}| \leq |\{d \in dist_p(\mathcal{X}, \mathcal{Y}) | d \leq \varepsilon\}|$: Lemma 7.2 states that the distances in $UB_p(\mathcal{X}_a, \mathcal{Y}_a)$ are upper bounds of

```
PBRQ($\mathcal{Q}$, $\varepsilon$, $\tau$, $\mathcal{D}$)
  $Q_{Ref}$ = queue containing all $\mathcal{X} \in \mathcal{D}$ ordered w.r.t.
        descending upper bounding probabilities;
  $result = \emptyset$;
  while $Q_{Ref} \neq \emptyset$ do
    $\mathcal{X}$ := $Q_{Ref}$.removeFirstElement();
    if $\Pr_{LB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon) \geq \tau$ then
      $result$.add($\mathcal{X}$);
    else if $\Pr_{UB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon) \geq \tau$ then
      refine $\mathcal{X}$;
      insert $\mathcal{X}$ into $Q_{Ref}$;
  return $result$;
```

Figure 7.7: Pseudocode of the PBRQ algorithm.

the corresponding exact distances in $dist_p(\mathcal{X}, \mathcal{Y})$. So the exact distances may be smaller. Consequently, more of the exact distances could be smaller than ε. Thus, the inequality holds.
(2) Analogously we can show that
$|\{d \in LB_p(\mathcal{X}_a, \mathcal{Y}_a) | d \leq \varepsilon\}| \geq |\{d \in dist_p(\mathcal{X}, \mathcal{Y}) | d \leq \varepsilon\}|$. □

7.5 Multi-Step Probabilistic Range Query Processing

In this section we outline how to use the above defined approximations and bounds to efficiently answer probabilistic bounded range queries and probabilistic ranked range queries.

7.5.1 Probabilistic Bounded Range Queries

The pseudocode of the PBRQ algorithm is given in Figure 7.7. Our query strategy follows an iterative filter-refinement policy.

A queue Q_{Ref} is used to organize all uncertain time series sorted by descending upper bounding probabilities. In an iterative process we remove the first element $\mathcal{X}$ of the queue Q_{ref}, compute its lower and upper bounding probabilities $\Pr_{LB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon)$ and $\Pr_{UB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon)$, and filter $\mathcal{X}$ according to these bounds. If $\Pr_{LB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon) \geq \tau$, then $\mathcal{X}$ is a true hit and is added to the result set. If $\Pr_{UB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon) < \tau$, then $\mathcal{X}$ is a true drop and can be pruned. Otherwise, $\mathcal{X}$ has to be refined. Let us note that we do not immediately refine the object completely. Rather, the refinement is performed in several steps, from 1^{st}-level to 2^{nd}-level approximations and finally to the exact representation. As mentioned above, not the complete refinement takes place in a single iteration. The time slots are rather refined separately. Details on the strategies for the step-wise refinement are presented in Section 7.5.3. After the partial refinement step, $\mathcal{X}$ is re-inserted into Q_{Ref} if it cannot be pruned or reported as true hit according to the above conditions and has not been refined completely. If an object $\mathcal{X}$ has been refined completely, then obviously $\Pr_{LB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon) = \Pr_{UB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon) = \Pr(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon)$. The iteration loop stops if Q_{Ref} is empty. In this case all objects have been pruned, identified as true hits before their complete refinement, or have been completely refined.

7.5.2 Probabilistic Ranked Range Query

The pseudocode for the PRRQ algorithm is presented in Figure 7.8. In a first step, a priority queue Q_{Rank} is initialized containing all time series $\mathcal{X} \in \mathcal{D}$ ordered by descending upper bounding probability $\Pr_{UB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon)$. After initialization, the method *getNext()* can be called, returning the next object in the ranking. Obviously, an object $\mathcal{X}$ is the object with the highest probability if for all objects $\mathcal{Y} \in \mathcal{D}$ the following property holds:

```
PRRQ($\mathcal{Q}$, $\varepsilon$, $\mathcal{D}$)
  /** Initialize Ranking **/
  $Q_{Rank}$ = queue containing all $\mathcal{X} \in \mathcal{D}$ ordered w.r.t.
        descending upper bounding probabilities;
  /** Method getNext() **/
  while $\Pr_{LB}(dist_p(\mathcal{Q}, Q_{Rank}$.elementAt(1)) $\leq \varepsilon) \leq$
        $\Pr_{UB}(dist_p(\mathcal{Q}, Q_{Rank}$.elementAt(2)) $\leq \varepsilon)$ do
    refine $Q_{Rank}$.elementAt(1);
    reorganize $Q_{Rank}$;
  $result$ = $Q_{Rank}$.removeFirstElement();
  return $result$;
```

FIGURE 7.8: Pseudocode of the PRRQ algorithm.

$\Pr_{LB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon) \geq \Pr_{UB}(dist_p(\mathcal{Q}, \mathcal{Y}) \leq \varepsilon)$. If the lower bound of the first object is already higher than the upper bound of the next object, than the exact probability value of the second object can not become larger than the exact probability value of the first object, and so the first object can be reported as the next object. Since the candidate objects of the database are ordered by descending upper bounding probabilities in Q_{Rank}, we only have to check if the lower bounding probability of the first element in Q_{Rank} is greater or equal to the upper bounding probability of the second element. If this test returns true, we can report the first object as the next ranked object. Otherwise, we have to refine the first object in Q_{Rank} in order to obtain better probability bounds. As discussed above, this refinement is step-wise, i.e. several refinement steps are necessary in order to obtain the exact probability or at least better upper and lower bounds. The idea of the method *getNext()* is to iteratively refine the first object in Q_{Rank} as long as the lower bounding probability of this element is lower than the upper bounding probability of the second element in Q_{Rank}.

7.5.3 Step-Wise Refinement of Probability Estimations

So far, we have not detailed how we refine our objects in the PBRQ and PRRQ algorithms. The aim of a refinement step is to refine the distances between a database object and the query object in order to get a better approximation (or the exact value) of the distances and thus, a better approximation of the probability. Since the refinement operations are very expensive, we propose to perform a step-wise refinement by trying to perform potentially cheap and rewarding refinement increments. Incremental refinements of an uncertain time series $\mathcal{X}$ are only performed as long as we cannot exactly determine whether $\mathcal{X}$ fulfills the query predicate.

Once, we have decided for which uncertain time series we want to execute the next refinement step (the top-ranked object in the refinement queue), there are several possibilities. Let us assume that we want to perform the refinement of the uncertain time series $\mathcal{X}$. There are multiple sample approximations of $\mathcal{X}$ that are potential refinement candidates. So we have to determine which of the candidates should be refined next. In the following, we present our refinement strategy.

We assume that the uncertain time series are organized in a priority queue which is sorted by descending upper bound probabilities (cf. Section 7.5.1 and Section 7.5.2). Furthermore we assume that the queue only contains those uncertain time series that currently can neither be assigned to the result set nor be identified as a true drop. In the example in Figure 7.9 there are depicted several probability intervals for different uncertain time series. In this example, the assumption holds for the uncertain time series $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{V}$ which are potential candidates for the next refinement step.

Refinement Goal

The aim for each refinement step is to be able to identify an uncertain time series as true hit or true drop. This is achieved for an uncertain time series $\mathcal{X}$ if the probability interval $[\Pr_{LB}(dist_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon), \Pr_{UB}(dist_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon)]$ is above or below τ. For this reason, we try to increase the lower bound of

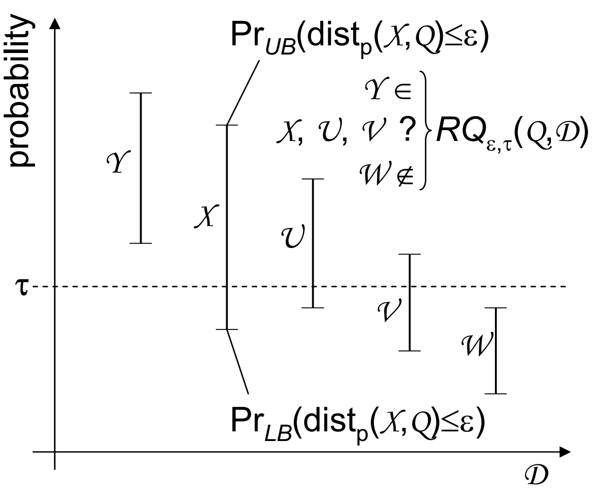

probability
$\Pr_{UB}(dist_p(\mathcal{X},\mathcal{Q})\leq\varepsilon)$
$\mathcal{Y}\in$
$\mathcal{X}, \mathcal{U}, \mathcal{V}$?
$\mathcal{W}\notin$
$RQ_{\varepsilon,\tau}(\mathcal{Q},\mathcal{D})$
$\mathcal{Y}$
$\mathcal{X}$
$\mathcal{U}$
τ
$\mathcal{V}$
$\mathcal{W}$
$\Pr_{LB}(dist_p(\mathcal{X},\mathcal{Q})\leq\varepsilon)$
$\mathcal{D}$

Figure 7.9: Probability intervals for different uncertain time series.

the probability $\Pr_{LB}(dist_p(\mathcal{Q},\mathcal{X})\leq\varepsilon)$ in case that

$$\tau - \Pr_{LB}(dist_p(\mathcal{Q},\mathcal{X})\leq\varepsilon) \leq \Pr_{UB}(dist_p(\mathcal{Q},\mathcal{X})\leq\varepsilon) - \tau$$

holds. Otherwise, we try to decrease $\Pr_{UB}(dist_p(\mathcal{Q},\mathcal{X})\leq\varepsilon)$. This technique tries to minimize the effort to decide whether an uncertain time series can be safely included in or excluded from the result set. We try to tighten the bound which is nearer to the crucial probability threshold. In the example shown in Figure 7.9, we would try to increase the lower bounds of the probabilities for $\mathcal{X}$ and $\mathcal{U}$. For $\mathcal{V}$ we would try to decrease the upper bound of the probability.

Refinement Strategy

Let us assume that we try to increase the lower bounds of the probabilities for $\mathcal{X}$. Then, we have to select an approximated distance that should be refined next. Keep in mind, an approximated distance corresponds to a pair of approximated regular time series. One element of the pair is an approximated regular time series of $\mathcal{X}$, the other element an approximated regular time series of $\mathcal{Q}$. In the following we will develop a criterion to determine which pair of approximated regular time series should be refined next.

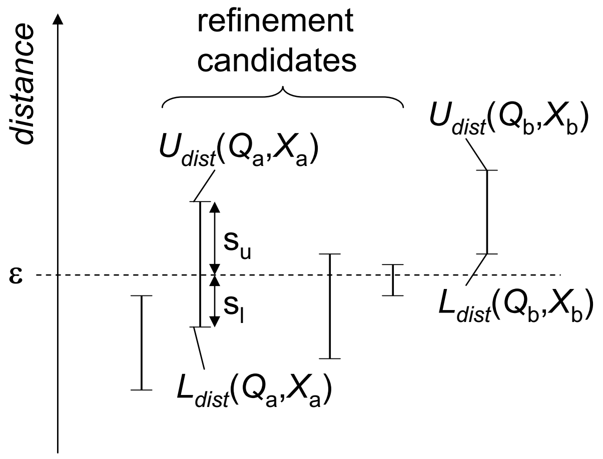


Figure 7.10: Distance intervals corresponding to pairs of approximated regular time series.

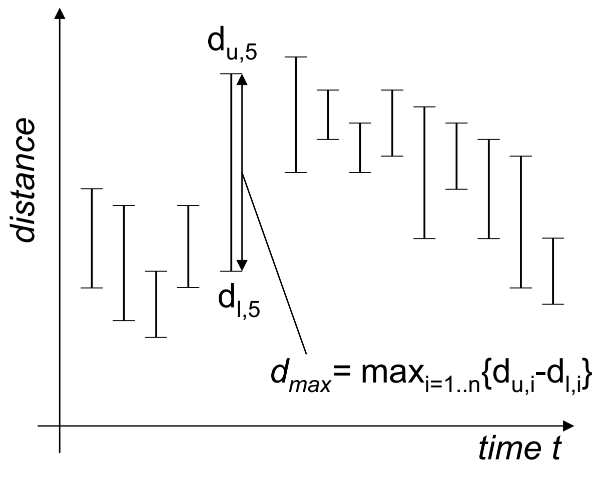


Figure 7.11: Contribution of different time slots to the distance estimation for a single pair of approximated regular time series.

In Figure 7.10 we depicted an example of several available pairs of approximated regular time series of $\mathcal{X}$ and $\mathcal{Q}$. In this figure, the resulting distance intervals are depicted. Potential candidates for the refinement are those distance intervals which are intersected by the ε value. For all other distance intervals, all represented exact distances are above or below ε, and no further insight would be gained by refining such intervals. In our example, the pair corresponding to X_a and Q_a is worth a refinement step, while the pair X_b and Q_b already represents only distances larger than ε.

The lower bound of the probability $\Pr_{LB}(dist_p(\mathcal{Q},\mathcal{X}) \leq \varepsilon)$ depends on the overall number of distances $d \in dist_p(\mathcal{Q},\mathcal{X})$ which are below ε. The higher this number, the higher the lower bound of the probability. Consequently, we should first refine that approximated distance (i.e. pair of approximated regular time series) which probably will be resolved into a set of approximated distances that are clearly below ε. Furthermore it should represent as many distances $d \in dist_p(\mathcal{Q},\mathcal{X})$ as possible. Here we use the following heuristic: The increase of the number of detected distances $d \in dist_p(\mathcal{Q},\mathcal{X})$ that are clearly below ε can be estimated by

$$\widetilde{w} = (1 - \frac{s_u}{\max_{i=1..n}\{d_{u,i} - d_{l,i}\}}) \cdot |X_a| \cdot |Q_a|,$$

s_u is the portion of the distance interval that is above ε (see Figure 7.10). Formally $s_u = U_{dist}(Q_a, X_a) - \varepsilon$. s_u is a measure for how easy it is to push the distances represented by this distance interval below ε. The smaller s_u, the more of the represented distance values are probably already below ε. Note we assume a uniform distribution of represented distance values.

After the decision which distance interval to refine next, we only refine the approximations at a single time slot. Therefore, the complete distance interval can only change so much as the maximal distance contribution over all considered time slots is. That is why we scan over all time slots and search for the maximal value of $(d_{u,i} - d_{l,i})$, where $d_{u,i} = \max\{u_{q_i} - l_{x_i}, u_{x_i} - l_{q_i}\}$ and $d_{l,i} = \max\{0, \max\{l_{q_i}, l_{x_i}\} - \max\{u_{q_i}, u_{x_i}\}\}$. $|X_a| \cdot |Q_a|$ corresponds to the number of distances which are approximated by $U_{dist}(Q_a, X_a)$ and $L_{dist}(Q_a, X_a)$. So, $\widetilde{w}$ reflects how probable it is, that the refined distance estimates will be smaller than ε.

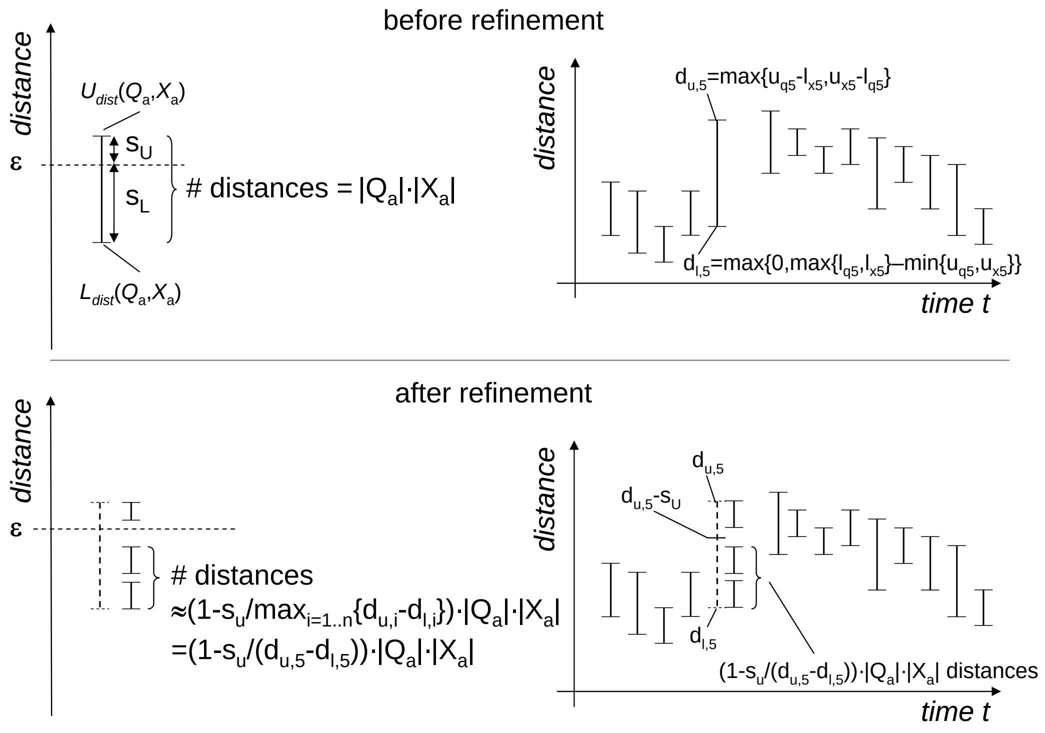

before refinement
distance
$U_{dist}(Q_a,X_a)$
s_U
s_L
ε
# distances = $|Q_a|\cdot|X_a|$
$L_{dist}(Q_a,X_a)$
$d_{u,5}=\max\{u_{q5}-l_{x5},u_{x5}-l_{q5}\}$
$d_{l,5}=\max\{0,\max\{l_{q5},l_{x5}\}-\min\{u_{q5},u_{x5}\}\}$
time t
after refinement
# distances
$\approx(1-s_u/\max_{i=1..n}\{d_{u,i}-d_{l,i}\})\cdot|Q_a|\cdot|X_a|$
$=(1-s_u/(d_{u,5}-d_{l,5}))\cdot|Q_a|\cdot|X_a|$
$d_{u,5}$
$d_{u,5}-s_U$
$d_{l,5}$
$(1-s_u/(d_{u,5}-d_{l,5}))\cdot|Q_a|\cdot|X_a|$ distances

Figure 7.12: Refinement heuristic.

The example depicted in Figure 7.12 motivates the heuristic. This example shows the situation of the approximated distance consisting of the pair Q_a and X_a before (top) and after (bottom) the refinement step. This distance can be bounded by $L_{dist}(Q_a, X_a)$ and $U_{dist}(Q_a, X_a)$. These bounds correspond to the aggregated distance bounds observed at each time slot. So, the remaining question is, which of the n distance intervals in the time domain should be refined. When refining such an interval in the time domain, like $(d_{l,5}, d_{u,5})$ in our example, all resulting distance intervals that are below $d_{u,i} - s_u$ correspond to the resulting approximated distances that are below ε. Since $\widetilde{w}$ has to be maximized, we should refine the largest time interval in the time domain. Finally, based on the described estimation, we refine the approximated distance for which $\widetilde{w}$ is maximal.

In case we decided to decrease the upper bound of the probability value, i.e. $\Pr_{UB}(dist_p(\mathcal{Q}, \mathcal{X}) \leq \varepsilon)$, we can adapt the above-described refinement strategy analogously. In this case, we have to maximize the overall number of distances $d \in dist_p(\mathcal{Q}, \mathcal{X})$ which are above ε. We can achieve this goal by simply replacing the parameter s_u with s_l when evaluating the approximated distance to be refined next. s_l is the portion of the distance interval that is below ε (see Figure 7.10). Formally $s_l = \varepsilon - L_{dist}(Q_a, X_a)$.

In general, we can refine either the database object or the query object or both of them. We propose to refine both in one refinement step. However, we refine the query object only virtually, i.e. for each time slot of the query time series, we have access to all approximation levels and the exact representation. In fact, the approximation level of the query object at time slot i will be adapted during the distance computation to the approximation level of the database object at time slot i. For example, if the approximation of $\mathcal{X}_a \in \mathcal{D}$ at time slot i is of level 2, we also consider the approximation of level 2 for $\mathcal{Q}_a$.

7.5.4 Probabilistic Queries Using DTW

The algorithms for PBRQ and PRRQ proposed so far are suitable for both the L_p-norms or DTW. However, if we use the DTW distance, we observe that the lower and upper bounding probabilities for $\Pr(dist_{DTW_p}(\mathcal{Q}, \mathcal{X}))$ are more expensive to calculate than for $\Pr(dist_{L_p}(\mathcal{Q}, \mathcal{X}))$, as the underlying distance measure is more expensive to calculate. However, we can adapt the algorithms for PBRQ and PRRQ in order to address this potential performance issue. The key idea for this adaption is that the exact L_p-distance is an upper bound for the DTW distance.

Lemma 7.4 (Lower Bound for the DTW-based Probability).

$$\Pr_{LB}(dist_{L_p}(\mathcal{Q}, \mathcal{X}) \leq \epsilon) \leq \Pr(dist_{L_p}(\mathcal{Q}, \mathcal{X}) \leq \epsilon) \leq \Pr(dist_{DTW_p}(\mathcal{Q}, \mathcal{X}))$$

Proof. *When calculating the probabilities* $\Pr_{L_p}$ *and* $\Pr_{DTW_p}$ *the number of distances* $\leq \varepsilon$ *has to be determined. Whenever the* L_p *distance is* $\leq \varepsilon$*, the corresponding DTW distance is also* $\leq \varepsilon$*, as the* L_p *distance is a valid solution of the DTW computation. Hence, the* L_p *distance is an upper bound* *for the DTW distance and therefore the number of DTW distances that are smaller than ε is equal to or larger than the number of L_p distances.* □

According to Lemma 7.4, we can use (the lower bound of) the probability based on the L_p-norm as an additional lower bound for the probability based on the DTW distance. If using DTW, the algorithms for PBRQ and PRRQ proposed above can be extended by using $\Pr_{LB}(dist_{L_p}(\mathcal{Q}, \mathcal{X}))$ as a first filter in order to identify true hits without calculating the bounds for the probability based on DTW.

PBRQ

In Figure 7.13 the pseudocode for the PBRQ for the DTW is depicted. The first condition checks whether the lower bound based on the L_p distance is larger than ε. This is a very efficient way to include true hits without calculating the exact DTW distances as the computation of the DTW distances is much more expensive than the computation of the L_p distances. $\Pr_{LB_{DTW}}(dist_p(\mathcal{Q}, \mathcal{X}))$ and $\Pr_{UB_{DTW}}(dist_p(\mathcal{Q}, \mathcal{X}))$ can then be used to prune true drops and to include further true hits, respectively.

PRRQ

The DTW version of the PRRQ (see Figure 7.13) is similar to its L_p version as well. The first element of the queue is refined until lower probability bound is larger than the upper probability bound of the second element in the queue. At first the $\Pr_{LB_{LP}}$ value is compared, as it is more efficient to compute.

7.6 Evaluation

In this section, we examine the efficiency of our proposed probabilistic similarity query approach for uncertain time series. Since the computation is

```
$PBRQ_{DTW}(\mathcal{Q}, \varepsilon, \tau, \mathcal{D})$
  $result = \emptyset$;
  $Q_{Ref}$ = queue containing all $\mathcal{X} \in \mathcal{D}$ ordered w.r.t. ascending refinement priority;
  while $Q_{Ref} \neq \emptyset$ do
    $\mathcal{X}$ := $Q_{Ref}$.elementAt(1);
    if $\Pr_{LB_{LP}}(\mathcal{X}) \geq \tau$ then
      $result$.add($\mathcal{X}$)
    else if $\Pr_{UB_{DTW}}(\mathcal{X}) < \tau$ then
      $Q_{Ref}$.remove($\mathcal{X}$); /** Prune $\mathcal{X}$ **/
    else if $\Pr_{LB_{DTW}}(\mathcal{X}) \geq \tau$ then
      $result$.add($\mathcal{X}$)
    else
      refine $\mathcal{X}$;
      update $Q_{Ref}$;
    end-if;
  end-while;
  return $result$;
```

Figure 7.13: Adaption of the PBRQ algorithm for the DTW.

highly CPU-bounded, we measured the efficiency by the average number of required calculations required to execute a query.

7.6.1 Datasets and Methods

We used benchmark datasets derived from a wide range, covering a broad spectrum of data characteristics. Most of them are available from the UCR Time Series Data Mining Archive [KF02] as described in Chapter 3. We varied the size of the audio dataset between 480 and 9600 time series, where all time series had a length of 100. The size of the CBF dataset was varied between 990 and 19800 time series by applying the creation process described in [Sai94].

Because all of the datasets contain exact measurements, we generated probabilistic time series by generating samples uniformly distributed around

```
PRRQ($\mathcal{Q}$, $\varepsilon$, $\mathcal{D}$)
  /** Initialize Ranking **/
  $Q_{Rank}$ = queue containing all $\mathcal{X} \in \mathcal{D}$ ordered w.r.t. descending $\Pr_{UB_{DTW}}$
  /** Method getNext() **/
  while ($\Pr_{LB_{LP}}(\mathcal{Q}, Q_{Rank}$.elementAt(1))<
          $\Pr_{UB_{DTW}}(\mathcal{Q}, Q_{Rank}$.elementAt(2))) AND
          ($\Pr_{LB_{DTW}}(\mathcal{Q}, Q_{Rank}$.elementAt(1))<
          $\Pr_{UB_{DTW}}(\mathcal{Q}, Q_{Rank}$.elementAt(2))) do
    refine $Q_{Rank}$.elementAt(1);
    reorganize $Q_{Rank}$;
  $result$ = $Q_{Rank}$.removeFirstElement();
  return $result$;
```

FIGURE 7.14: Adaption of the PRRQ algorithm for the DTW.

the given exact values. We also used other distributions like the Gaussian distribution, but since our experiments showed that the distribution of the samples do not make a difference in for the general results, we only report the results using uniform sample distributions.

To support this decision we counted the mathematical operations required to answer different probabilistic bounded range queries on two datasets. The first dataset was created by sampling 6 points according to the Gaussian distribution, the second dataset was obtained by sampling 6 points according to the uniform distribution. Although these dataset resemble each other, the actual distance sets differ, and so it is no surprise that the exact numbers of required operations differ slightly. The results are depicted in Figure 7.15. All example queries on both datasets require between 2.35 million and 2.45 million operations. Even if the sample range around the original exact points is increased, or if other τ or ε parameters are used, the difference between the dataset generation methods is still very small compared to the number of required calculations of the straightforward approach. In this case this number is as high as 8.6×10^{97}. This follows from Lemma 7.1 which calculates the number of possible distance values. In the following, we assume

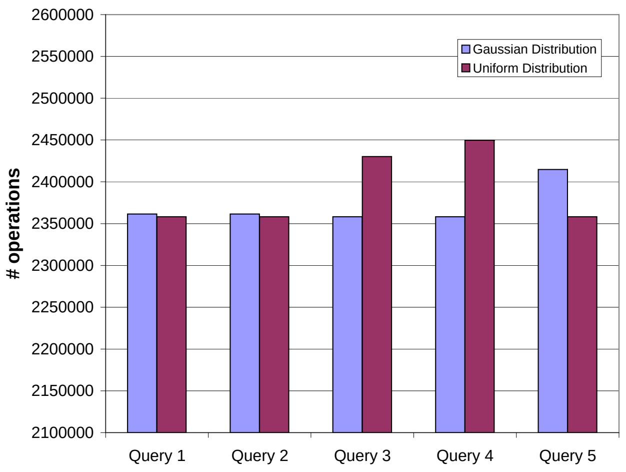


FIGURE 7.15: CPU cost w.r.t. different dataset generation methods.

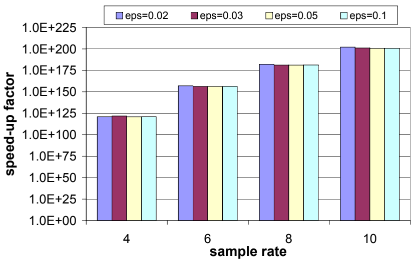
that in order to compute one single distance value, n (the number of time slots) basic operations have to be performed. So, a basic operation includes the calculation of the absolute value between two points and the power of p calculation. The root operation can be neglected as it occurs very rarely compared to the huge amount of other calculations.

7.6.2 Experimental Results

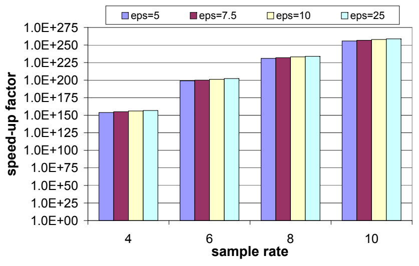
At first we measured the speed-up factor our approach yields compared to the straightforward approach. Afterwards we explored how well our approach scales with respect to database size and time series length. Finally, we examined the impact of our refinement strategy on the observed speed-up.

Overall Speed-Up

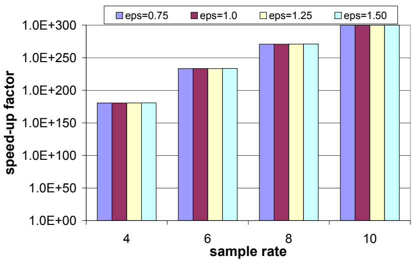
For our experiments we measured the amount of required basic calculations to execute the corresponding queries as explained above. Based on these numbers, we define the speed-up factor as the ratio between the required calculations of the straightforward approach and the required operations using our approach.



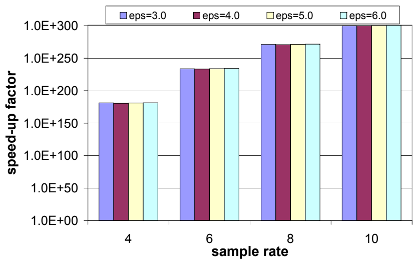
(a) Audio (size = 9600)



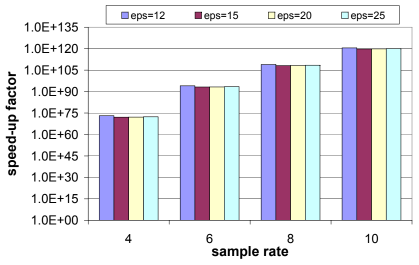
(b) CBF (size = 19800)



(c) GunX



(d) Leaf

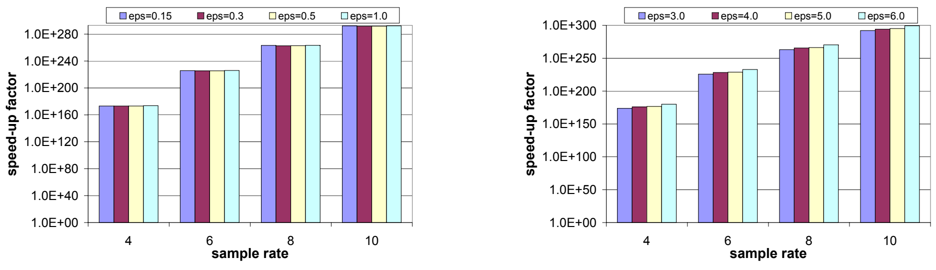


(e) SynCtrl

Figure 7.16: Query speed-up (PBRQ) for different datasets and varying sample rates.

In the first experiment, we examined how our approach can speed up probabilistic bounded range queries for different datasets. We repeated the experiment for varying sample rates. The results for the different datasets are shown in Figure 7.16. For all datasets, the speed-up factor increases exponentially with linearly increasing sample rate. The reason for this observation is that the cost required for the straightforward query method increases exponentially while our pruning strategies work well even for high sample rates. At the same time we varied the ϵ parameter. As can be seen in the same figure, different choices for the ϵ value lead to different values for the computational cost. This is due to the different size of the result set of the corresponding query. Furthermore, different ϵ values lead to different pruning possibilities and hence, different refinement processes take place. So, depending on the given ϵ, different number of refinement steps are necessary.

In the next experiment, the above mentioned experiments were repeated for probabilistic ranked range queries. The results are depicted in Figure 7.17 and are very similar to those for the probabilistic bounded range queries. We depict the results for only two datasets, as the other datasets showed the same behavior.



(a) GunX (b) Leaf

FIGURE 7.17: Query speed-up (PRRQ) for different datasets and varying sample rates.

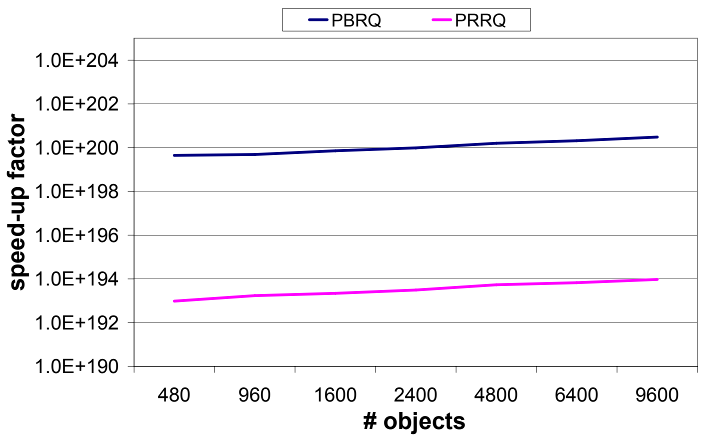
Scalability

Next, we examined the scalability of our approach with respect to database size and time series length. For the first experiment we varied the database size of the Audio dataset between 480 and 9600 time series objects. The size of the CBF dataset was varied between 990 and 19800 elements. In Figure 7.18 the observed speed-up factors for probabilistic bounded range queries as well as for probabilistic ranked range queries are depicted.

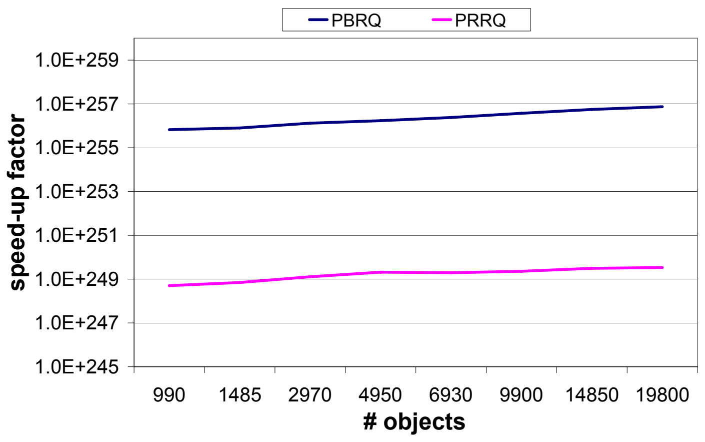
The next experiment analyzes the impact of the time series length. In order to compare similar datasets with different time series length, we extended the time series of the SynCtrl dataset by copying earlier portions of a time series to the end of the time series. That way we created datasets with time series of length 50, 75, 100, 125, and 150 time slots. The observed speed-up factors are shown in Figure 7.19. Note that this figure is presented in logarithmic scale as the speed-up factor increases very fast for longer time series. This is due to the fact that the length of the time series has an exponential impact on the number of possible distance values and so our approach can save a larger amount of computations. In contrast, the size of a dataset has only a linear impact on the required calculations for the naive approach, so the observed speed-up is lower.

Number of Approximations

In the next experiment we analyzed the impact of the number of second-level approximations. As explained in Section 7.4, we use the k-means algorithm to cluster the sample points in order to create the approximations. We performed the experiments for 10 and for 20 sample points using the SynCtrl dataset. The results in terms of required basic calculations are presented in Figure 7.20. Although the number of sample points was twice as high in the second run of the experiment, the optimal choice for the number of second-level approximations was the same as for the first run. A similar behavior was observed on all datasets and for further numbers of sample points. We learned that relatively low values for the k parameter in k-means yield



(a) Audio (sample rate = 10, #level 2 approx. = 3)



(b) CBF (sample rate = 10, #level 2 approx. = 3)

Figure 7.18: Scalability of probabilistic bounded range queries and probabilistic ranked range queries with respect to the database size.

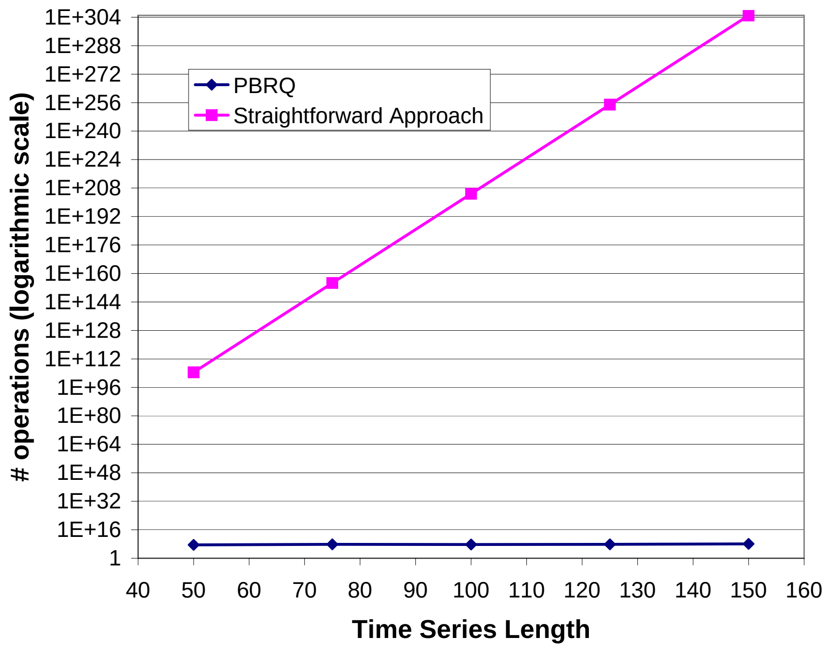


FIGURE 7.19: Scalability of PBRQ with respect to the time series length (SynCtrl, sample rate = 10, #level 2 approx. = 3). Logarithmic Scale.

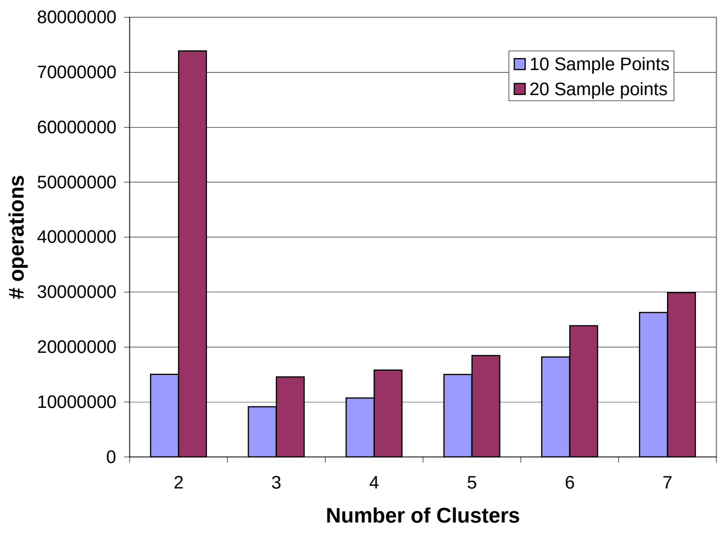


FIGURE 7.20: Number of second-level approximations (clusters) vs. number of required calculations. SynCtrl dataset, PBRQ.

good results. This is due to the fact, that the number of representations exponentially influences the required amount of calculations. However, if the approximations are too coarse (for very low values of k), it is very probable that a large portion of all approximations have to be refined and subsequently not very much computational effort can be saved.

Refinement Strategy

The above demonstrated huge speed-up factors compared to the straightforward approach can be observed for a broad range of parameter settings, different datasets, and different query types. In order to assess the impact of our refinement strategy, in this section we report the absolute values of required calculations for different refinement strategies, rather than comparing them to the straightforward approach.

Our refinement strategy actually consists of two steps. First, a pair of regular approximated time series is chosen, second, the point in time at which to refine is determined. This leads to 4 different refinement strategies: First, for both steps we apply the strategy described in Section 7.5.3. We denote this combined approach as 'S-S', as we use our strategy for both steps. Second, it is possible to randomly select a pair of regular approximated time series but use our refinement strategy to determine the point in time at which to refine. This approach is labeled by 'R-S'. Analogously the third strategy is called 'S-R', as the refinement strategy is used for the first step, but not for the second. Finally 'R-R' denotes a random choice for both steps. As we have shown above, the results for the different datasets are very similar, so in this section we restrict the presentation of our results to the SynCtrl dataset. In fact, the results on the other datasets are very similar.

At first we examined the behavior of the different strategies with respect to the ϵ parameter. The results are depicted in Figure 7.21. The second step, i.e. determining the correct point in time at which to refine is apparently more crucial, as 'R-S' significantly outperforms 'S-R'. As expected, using our refinement heuristic for both steps yields the best results. These observations can also be made in Figure 7.22 where we varied the probability bound τ

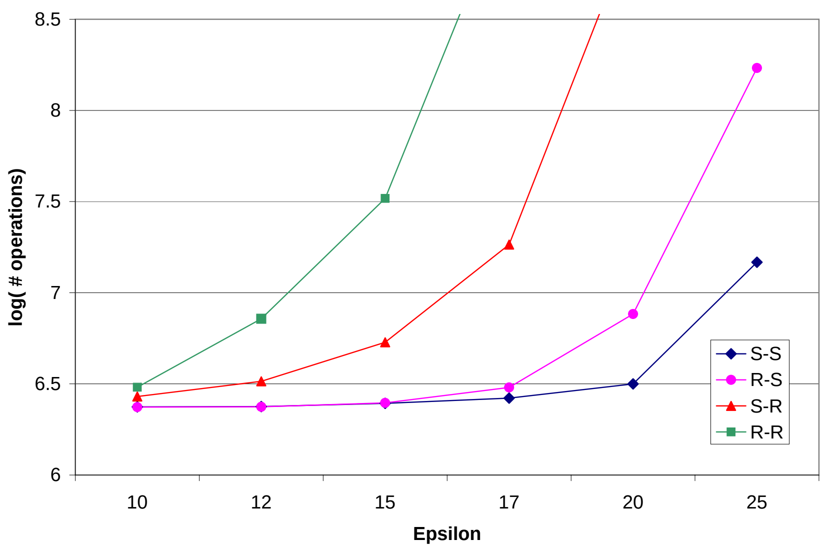


FIGURE 7.21: Required calculations (logarithmic scale) of different refinement strategies w.r.t. ϵ (dataset=SynCtrl, #level 2 approx.=2, sample rate=6, τ=0.95).

and in Figure 7.23 where we varied the sample rate. The differences between the different refinement strategies in general and the superiority of the 'S-S' approach in particular become more distinct when the total computational cost is higher. This is for example the case for a lower probability bound or a higher sample rate. In summary, our experiments show that our refinement strategy clearly outperforms more simple refinement strategies and that this superiority of our approach is robust w.r.t. all query parameters.

7.7 Conclusions

Similarity search on uncertain time series is an important emerging topic. To the best of our knowledge, we proposed the first approach for performing probabilistic similarity search on uncertain time series in this work. In particular, we formalized the notion of uncertain time series and introduced two novel probabilistic query types for uncertain time series. Furthermore, we proposed an original method for efficiently supporting these probabilis-

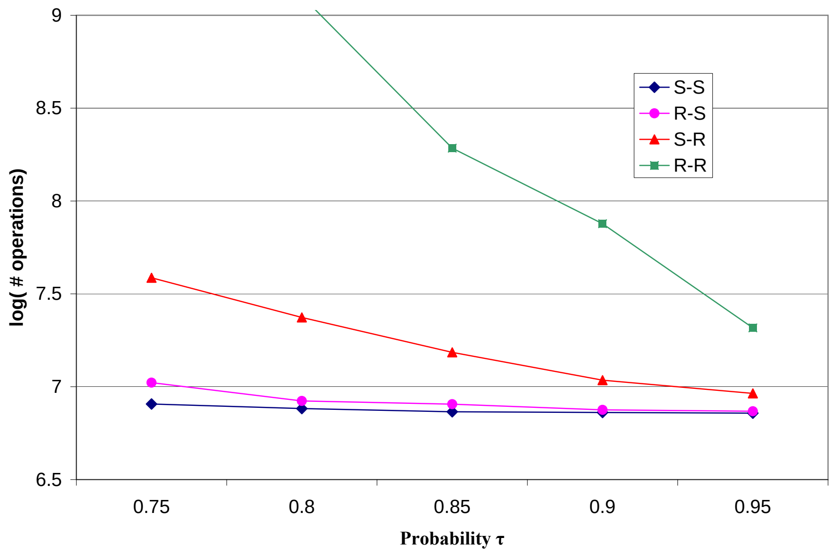


FIGURE 7.22: Required calculations (logarithmic scale) of different refinement strategies w.r.t probability τ (dataset=SynCtrl, #level 2 approx.=2, sample rate=6, ϵ=12).

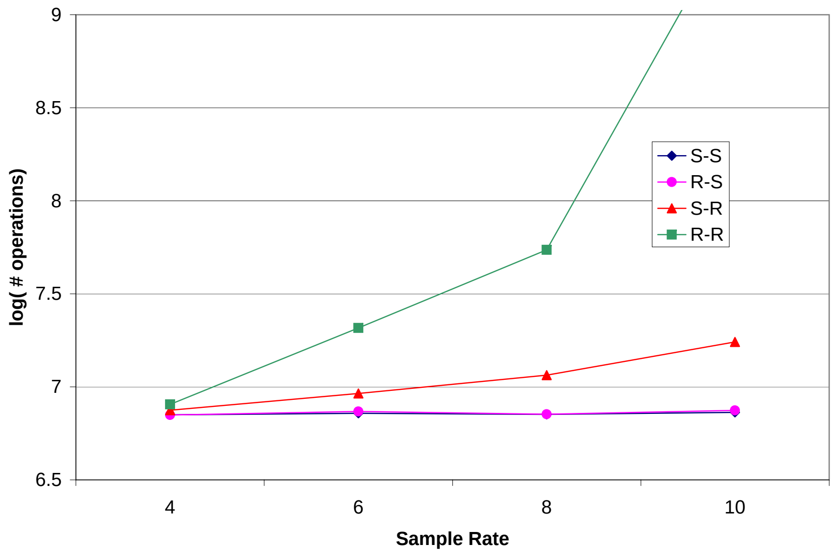


FIGURE 7.23: Required calculations (logarithmic scale) of different refinement strategies w.r.t sample rate (dataset=SynCtrl, #level 2 approx.=2, ϵ=12, τ=0.95).

tic queries using a filter-refinement query processing technique based on an approximative representation of uncertain time series and a sophisticated refinement strategy. Our experimental evaluation illustrates the performance gain of our method compared to competing strategies.

Part III

Data Mining on Temporal Data

Chapter 8

Semi-Supervised Threshold Queries

Threshold-based similarity as introduced in Chapter 4 has a great practical impact on a lot of application fields, including stock market analysis, astronomy, environmental analysis, molecular biology, and pharmacogenomics.

A sample application from medical analysis is visualized in Figure 8.1 where three electrocardiogram (ECG) plots $T1$, $T2$, and $T3$ are shown. Plot $T1$ indicates a high risk for cardiac infarct due to the abnormal deflection after the systole (ST-T-phase), whereas $T2$ and $T3$ both exhibit a normal behavior indicating a low risk. For the examination of time series with respect to this abnormal characteristic, there is no need to examine the complete curve and the exact recorded values. A better way to detect such kind of characteristics is to analyze only the relevant parts of the time series, for instance observing those parts of the time series which exceed a specified threshold as depicted in our example. Let us consider the time interval sequences (below the ECG-curves) which correspond to the time frames within which the time series exceed the threshold τ. We can observe that the time interval sequences derived from $T2$ and $T3$ differs marginally. In contrast, time series $T1$ shows quite a different characteristic, caused by the ECG-aberration indicating the heart disease.

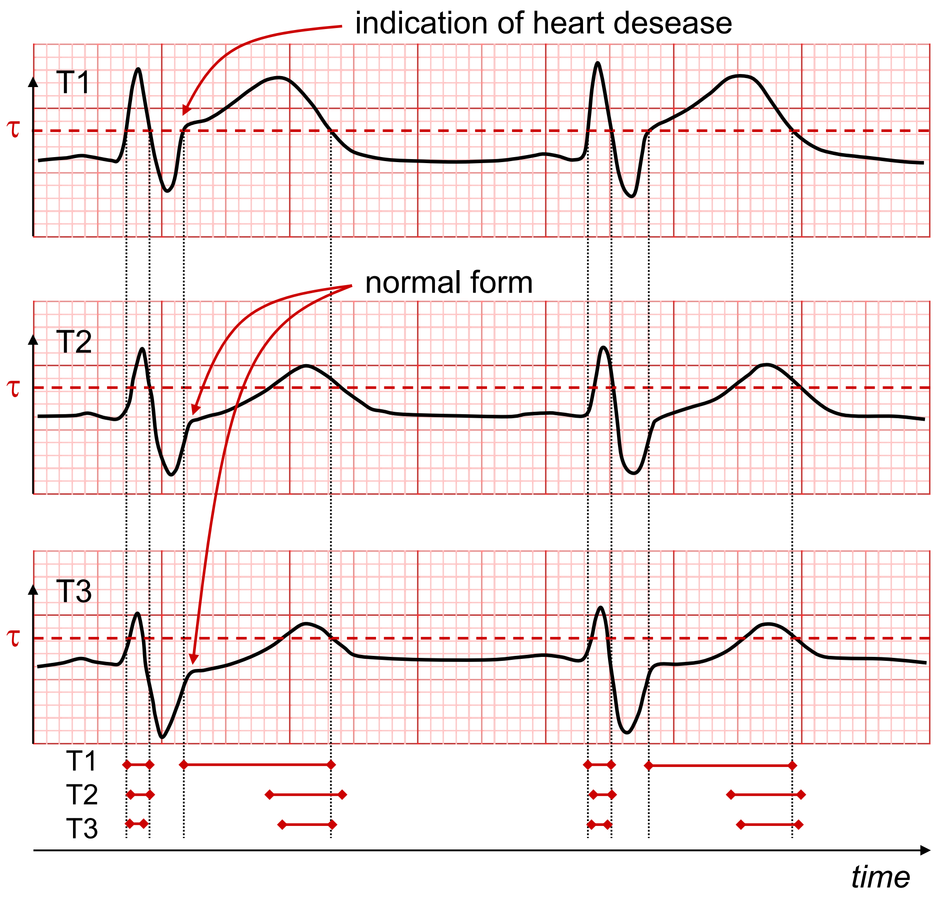


Figure 8.1: Sample application for the threshold-based similarity.

The most important issue of threshold similarity is obviously the choice of the threshold τ. In the medical example, the suitable threshold τ was selected by a domain expert (knowing about the characteristics of an abnormal time curve in case of a cardiac infarct patient), in order to discriminate between patients with a low or a high risk for cardiac infarct. However, it can be easily seen that if the threshold would have been chosen lower than depicted in Figure 8.1, all three time series would have produced rather similar time intervals and, thus, the time series $T1$ could not have been discriminated from the other two time series. Since the optimal threshold for discriminating a predefined class system is not known in advance in many applications, a method for the automatic determination of the optimal threshold using a small number of labeled time series as training set is mandatory. Thus,

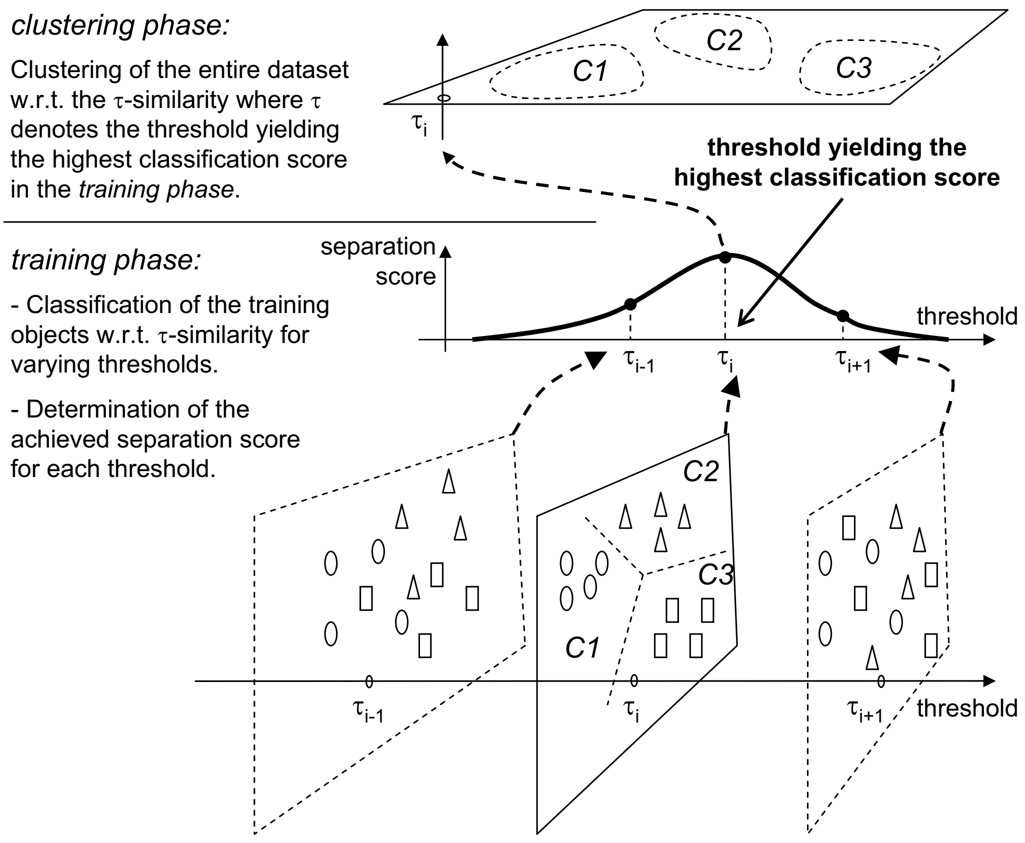

clustering phase:
Clustering of the entire dataset w.r.t. the τ-similarity where τ denotes the threshold yielding the highest classification score in the training phase.
C1
C2
C3
τ_i
threshold yielding the highest classification score
training phase:
- Classification of the training objects w.r.t. τ-similarity for varying thresholds.
- Determination of the achieved separation score for each threshold.
separation score
threshold
τ_{i-1}
τ_i
τ_{i+1}
C1
C2
C3
threshold

FIGURE 8.2: General approach.

for cluster analysis, a semi-supervised approach is envisioned where first, the best suitable threshold is determined automatically by means of a small training set. Afterwards a cluster analysis based on the threshold-based similarity measure can be performed using the previously learned threshold. This approach allows to detect novel and potentially important patterns.

In this chapter, we present a general semi-supervised framework for the cluster analysis of time series using adaptable threshold similarity. This framework consists of two phases, a training phase and a clustering phase as depicted in Figure 8.2. In the first phase, the most suitable parameter setting, i.e. the choice of the threshold value, is determined by using a training dataset. Our proposed method assumes that we can observe different clustering results for different threshold values. So, τ influences the separability of the classes which we quantify by a so-called separation score. First, we compute the separation score for each threshold of a given training set. This results in a quality curve depending on τ. The optima of this curve can give useful hints on how to adapt the threshold for the second phase where the entire dataset is clustered. One might argue that performing a number of clusterings for different thresholds followed by an evaluation of the clustering quality can lead to the same results. So, the training phase could be omitted. However, this ignores the fact that many clustering algorithms have a runtime of $O(n^2)$ in the number of objects to cluster, or require several iterations until they terminate. Furthermore we will show that our approach can handle missing classes as well. The first step should only indicate promising thresholds and avoid overfitting by not restricting the subsequent steps to a single threshold value.

We explored two different ways of determining such crucial thresholds. The first method outlined in Section 8.2.2 was introduced in [AKK$^+$06a] and uses the well-known silhouette coefficient in order to explore suitable threshold values. The second method described in Section 8.2.2 was introduced in [AKK$^+$06d] and uses kernel functions in order to compute separations scores. In Section 8.3 we show the results of an experimental comparison of both approaches.

8.1 Semi-Supervised Clustering

In addition to the similarity information used by unsupervised clustering approaches, in many cases a small amount of knowledge is available concerning either pairwise (must-link or cannot-link) constraints between data items or class labels for some items. In contrast to standard clustering techniques which do not use any knowledge except for the similarity information of the data, semi-supervised analysis can profit from this knowledge to guide or adjust the clustering. Obviously, semi-supervised analysis methods can achieve better results than their unsupervised counterparts. In recent years, several methods in the area of semi-supervised cluster analysis have been proposed.

The main idea of semi-supervised clustering is to determine clusters that are 'immaculate' with respect to the class labels. The labeled data is used as a feedback in order to help to cluster unlabeled data. Most of the proposed methods for semi-supervised clustering assume that class labels for all objects to be processed are given.

[SCSS05] proposes a method based on a mixture of hidden Markov models using prior knowledge in order to improve the robustness and the quality of the clustering. The authors of [Zho05] introduce a semi-supervised classification for time sequences based on hidden Markov models. Two different semi-supervised learning paradigms are discussed. The author observed that using unlabeled data can increase the classification accuracy.

Several extensions of existing standard clustering algorithms have been proposed in the literature. A brief survey is given in [EZZ04] describing SPAM a supervised variant of PAM, SRIDHCR, a greedy algorithm with random restart, SCEC, an evolutionary algorithm, TDS, a medoid-based top-down partitioning algorithm. In [WCRS01], a variant of a k-means-based clustering algorithm is proposed. The authors derive constraints from the labeled objects which are used during the clustering. They distinguish between explicit and cannot-link constraints. In [SBM04], a k-means based method is introduced which is based on both types of constraints and which exploits the data distribution. The authors of [DBE99] describe an evolutionary method for semi-supervised clustering. This approach has to be initialized with k arbitrary centroids and optimizes a quality measure considering cluster dispersion and impurity. In order to detect a cluster structure that reflects the class distribution of the labeled training data, further methods have been developed which use a standard clustering algorithm by applying an adaptive similarity measure. The authors of [KKM02] propose to apply a complete-link clustering algorithm after replacing the Euclidean distance with the shortest path algorithm. The approach described in [BM03] weights the edit distance using an expectation maximization algorithm to detect approximately duplicate objects in a database. [BSM04] describes a probabilistic framework for semi-supervised clustering to additionally support several non-Euclidian distance measures like the cosine distance.

In the following, we use the density-based hierarchical clustering algorithm OPTICS [ABKS99]. However, any clustering algorithm is applicable in our framework.

8.2 Semi-supervised Threshold Analysis

8.2.1 General Idea

As stated above, our main goal is to yield an accurate clustering of the database $\mathcal{D}$ of time series using the threshold-based similarity. In order to choose the optimal threshold value τ, we apply a semi-supervised clustering procedure, where we learn the optimal threshold from a small training set $\mathcal{T}$ of already labeled time series before clustering the complete dataset (see Figure 8.2). This learning phase preceding the clustering phase is the key step in our framework.

Let $\mathcal{T}$ be the training set containing time series objects that are labeled according to a predefined class system $\mathcal{C} = \{C_1, \ldots C_k\}$ of $k \geq 2$ classes. We need to learn the threshold values from the objects in $\mathcal{T}$ that are able to separate time series data of one training class C_i from the other training classes C_j $(i \neq j)$, i.e. threshold values that yield low similarity values for time series belonging to different classes and high similarity values for time series belonging to the same class.

The class system $\mathcal{C}$ defined for the training data $\mathcal{T}$ has not to be complete. There may be some further classes $\hat{C} \notin \mathcal{C}$ for which no training data is at hand, i.e. none of the objects in $\mathcal{T}$ is labeled with one of these classes $\hat{C}$. Furthermore, it is also possible that the user is not aware of the existence of all classes. The dataset may contain unknown classes which could also be interesting for the user.

Obviously, these classes are excluded from the learning phase, i.e. the learned threshold are not necessarily optimal for these classes. However, as we show in the experimental Section 8.3, threshold values that exhibit a high separability for only a few classes in the training data are quite often also a good choice for the detection of unknown or missing classes during the clustering phase. Thus, by providing only partial information during the training phase, our approach is able to retrieve novel information in the clustering phase. This is contrary to a fully supervised approach, where novel classes cannot be detected.

As mentioned above, the optimal threshold τ_{opt} separates the classes $C_i \in \mathcal{C}$, $1 \leq i \leq k$, in our training set $\mathcal{T}$ in a best possible way. We formalize the separability of a threshold τ by means of a *separation score*. In fact, we measure the separation score of a broad range of possible thresholds.

8.2.2 Computing the Separation Score

Using the Silhouette Coefficient

In this section we will outline how we calculate the separation score based on the silhouette coefficient.

Separation Score for a Single Class Let $\mathcal{T}$ be a training set, and $\mathcal{C}$ be the corresponding classification system. Let $\mathcal{C}$ consist of k classes, $\mathcal{C} = C_1, ..., C_k$. Each class C_i is a subset of $\mathcal{T}$. Then for a given class C_i, we try to determine these threshold values which yield a good separability of C_i from the remaining classes. For a given threshold τ, we compute the silhouette width [KR90] for C_i compared to the remaining classes of $\mathcal{C}$.

Definition 8.1 (Silhouette Width for Class C_i).
Let $X \in C_i$. Then $a(X)$ is the average threshold-based distance value of X to all other elements of C_i

$$a(X) = \frac{1}{|C_i| - 1} \sum_{Y \in (C_i \setminus X)} d_{TS}(S_X, S_Y)$$

where d_{TS} is the threshold-based similarity as defined in Chapter 4. For each class $C_j \in (\mathcal{C} \setminus C_i)$, $d(X, C_j)$ is the average threshold-based similarity of X

to all elements of C_j

$$d(X, C_j) = \frac{1}{|C_j| - 1} \sum_{Y \in (C_j \setminus X)} d_{TS}(S_X, S_Y)$$

After we have computed all values $d(X, C_j)$, we select the smallest one as $b(X)$.

$$b(X) = \min_{C_j \in (\mathcal{C} \setminus C_i)} d(X, C_j)$$

This value can be considered as the average distance of X to the nearest cluster that is not equal to the cluster, X itself is assigned to. The silhouette value $s(X)$ for X is defined as

$$s(X) = \frac{b(X) - a(X)}{max\{a(X), b(X)\}}$$

Finally, the silhouette width $S(C_i)$ for a cluster C_i can be calculated as the average silhouette value over all cluster members

$$S(C_i) = \frac{1}{|C_i|} \sum_{X \in C_i} s(X)$$

Separation Score for All Classes In the last section, we have developed a quality measure which computes the separation score for each class C_i of the training dataset. Now, we need a suitable combination of all k separation score functions. For our approach, we use the sum of all score functions, i.e. we compute the silhouette width for $\mathcal{C}$. The global separation score function now reflects the overall separability score of our training dataset for an arbitrary threshold τ. Based on the idea of semi-supervised learning, the global score function gives the user hints about the most promising threshold values.

Kernel-Based Approach

In this section, we compute an optimal score by applying the concept of support vector machines (SVMs). In general, SVMs provide an optimal separation of two classes [Vap95] and can easily be extended to multi-class problems. However, in order to apply SVMs to threshold similarity of time series we have to extend the basic concepts of threshold similarity. In Section 8.2.2 we first explain these extensions of threshold similarity. We then introduce a new separation score in order to measure how good a given threshold separates the class classes of $\mathcal{T}$ (see Section 8.2.2).

Similarity Model As discussed above, we use SVMs to separate the classes in $\mathcal{T}$ since they provide an optimal separation. However, basic SVMs can only be applied to feature vectors rather than interval sequences. Thus, in order to apply SVMs to time series using threshold similarity, we need the concept of kernel methods which have been successfully applied to learning from objects having a complex structure. Since, we represent the time series by sets of intervals we need a kernel method that can cope with set-based instances. Let χ denote the complete set of intervals of all objects in $\mathcal{T}$ generated by a given threshold τ. For our approach, in order to compare two time series $X, Y \in \mathcal{T}$, we use the set kernel $k(S_{\tau,X}, S_{\tau,Y})$ which has been introduced in [GFKS02] and is defined as

$$k(S_{\tau,X}, S_{\tau,Y}) := \sum_{i \in S_{\tau,X}, j \in S_{\tau,Y}} \kappa_\chi(i,j),$$

where κ_χ denotes a kernel on χ, i.e. on single intervals. The threshold-crossing intervals sets $S_{\tau,X}$ and $S_{\tau,Y}$ are defined according to Definition 4.1.

In order to keep the similarity function invariant to the size of the sets $S_{\tau,X}$ and $S_{\tau,Y}$, the kernel function has to be normalized by the cardinality of these sets:

$$\overline{k}(S_{\tau,X}, S_{\tau,Y}) := \frac{k(S_{\tau,X}, S_{\tau,Y})}{N(S_{\tau,X}) \cdot N(S_{\tau,Y})}.$$

Here, the normalization function

$$N(S) := \sum_{s \in S} S(s)$$

is used to compute the cardinality of the set S, i.e. $S(s)$ returns 1, if s is in the set S and 0 otherwise. Note, that this kind of normalization preserves the kernel property, i.e. the resulting function $\overline{k}_{set}$ is also a kernel [GFKS02].

The kernel function κ_χ is applied to a pair of threshold crossing time intervals and corresponds to the similarity of two intervals. Since the distance function d_{int} (see Definition 4.2) is not a kernel, we cannot apply it directly to κ_χ. Rather, we need a similarity function κ_χ which fulfills the kernel property. In addition, this similarity function should fulfill the following condition: the smaller the distance between two intervals, the higher the similarity between them. With this condition, the similarity between two time intervals depends on their temporal offsets of their starting and ending times. Intuitively, the closer two time intervals start and the closer their interval length, the more similar they are. In our approach we apply the Gaussian kernel, which is defined as follows:

$$\kappa_\chi(i,j) := e^{\frac{-d_{int}(i,j)^2}{\sigma}},$$

where σ is a parameter which can be used to adjust the sensitivity of the similarity to the distance between the intervals i and j. For small σ values, large interval distances have only little influence on the similarity.

Thus, the resulting kernel function to compare time series X and Y using the concept of threshold similarity w.r.t. threshold τ is defined as:

$$\mathcal{K}_\tau(X,Y) = \frac{\sum_{i \in S_{\tau,X}, j \in S_{\tau,Y}} e^{\frac{-d_{int}(i,j)^2}{\sigma}}}{N(S_{\tau,X}) \cdot N(S_{\tau,Y})}.$$

Let us note, that the similarity between two time series according to threshold τ expressed by the kernel function $\mathcal{K}_\tau$ is also called *τ-similarity.*

Kernel-Based Separation Score As we have defined the similarity function $\mathcal{K}_\tau$ as a kernel function, we can now apply the concept of SVMs in order to measure the separability of given training data with respect to a threshold τ. The use of SVMs provides an optimal solution for the separation of the classes in $\mathcal{C}$. In addition, SVMs already contain information about the separability of the training data with respect to the class labels and thus, provide an elegant method to measure the separability of a given class system.

At first, we have to determine those threshold values which could be of interest and which we want to examine. Therefore, we select a range of am-

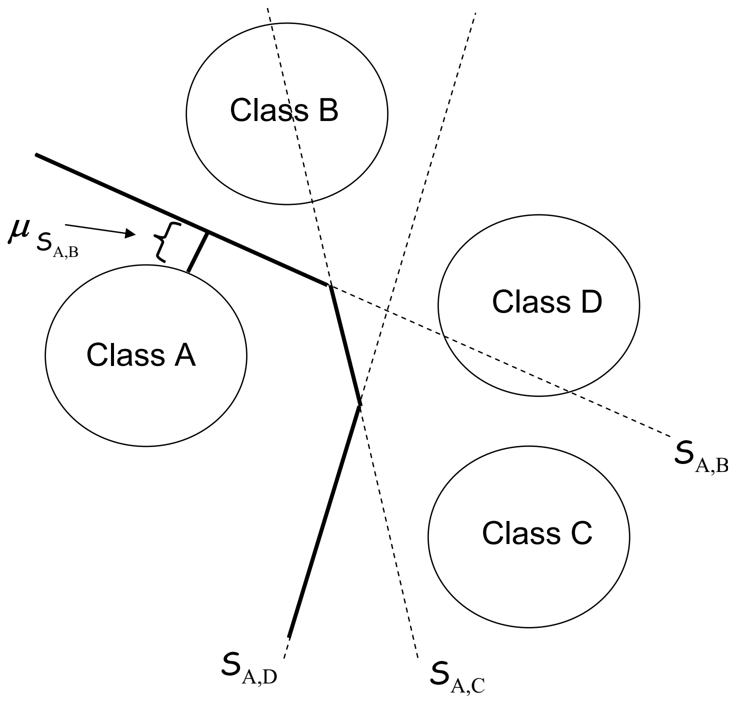


Figure 8.3: Computing the separation score.

plitudes which could be meaningful for our analysis. In our experiments, we have chosen the complete amplitude spectrum covered by the time series objects contained in the training dataset. However, if domain knowledge is available, this range of meaningful thresholds can be narrowed down. In addition, we can apply the data structures proposed in Chapter 4 to access the threshold-crossing time intervals of a given time series X for any threshold τ very efficiently. Afterwards, we have to choose the resolution of our examination, i.e. how many thresholds we want to examine within the selected range.

After we have selected the increments of the threshold values, we evaluate each threshold value τ as follows: We determine the threshold-crossing time interval sequences of all training objects w.r.t. τ and train an SVM on this data. Standard SVMs are able to make only binary decisions. An SVM $\mathcal{S}^{\tau}_{A,B}$ computes a maximum-margin hyperplane which separates instances of two classes A and B using the kernel function $\mathcal{K}_{\tau}$. The width of the margin $\mu^{\tau}_{A,B}$

of this separating hyperplane is a valid indication for the separability of the two classes. Obviously, the larger this width is, the more confident is the SVM to separate objects from A and B correctly. Since usually, $\mathcal{C}$ consists of more than two classes, and since standard SVMs can only handle the binary case, we apply the so-called *one-versus-one approach.* For each pair of classes $C_i, C_j \in \mathcal{C}$, a SVM $\mathcal{S}^{\tau}_{i,j}$ is trained. Thus, we obtain for each pair of classes $C_i, C_j \in \mathcal{C}$ the margin-width $\mu^{\tau}_{i,j} = \mu^{\tau}_{j,i}$ which is a measurement for the separability of C_i and C_j. An example is depicted in Figure 8.3. Four classes A, B, C, and D are separated using an SVM for each pair. Only the SVMs separating A from the other classes are shown. The width of the margin of $\mathcal{S}_{A,B}$, $\mu_{A,B}$ is visualized .

Given the margin-width of all SVMs trained on each pair $C_i, C_j \in \mathcal{C}$, all these values have to be combined suitably in order to obtain a single value which represents the global class separability. It is possible to represent the global separability by the smallest of all margin-widths. This approach guarantees that all desired classes are well separated. However, this solution seems to be a too optimistic approach, since the training dataset may contain classes which cannot be separated at all, regardless of the selected threshold value. Another approach is to pick the largest margin-width. However, this pessimistic solution is also not suitable, since two classes which are separable well for each threshold value do not reflect the global separability. For our approach we argue to use the average margin-width, because the separabilities of all observed classes are considered. This decision is confirmed by the results in our experiments. The resulting *separation score* is thus defined as

$$\text{score}(\tau) = \frac{1}{2 \cdot N(\mathcal{C})} \sum_{C_i, C_j \in \mathcal{C}, i \neq j} \mu^{\tau}_{j,i}.$$

Obviously, the higher this value is, the better the classes in $\mathcal{C}$ can be separated with respect to threshold τ.

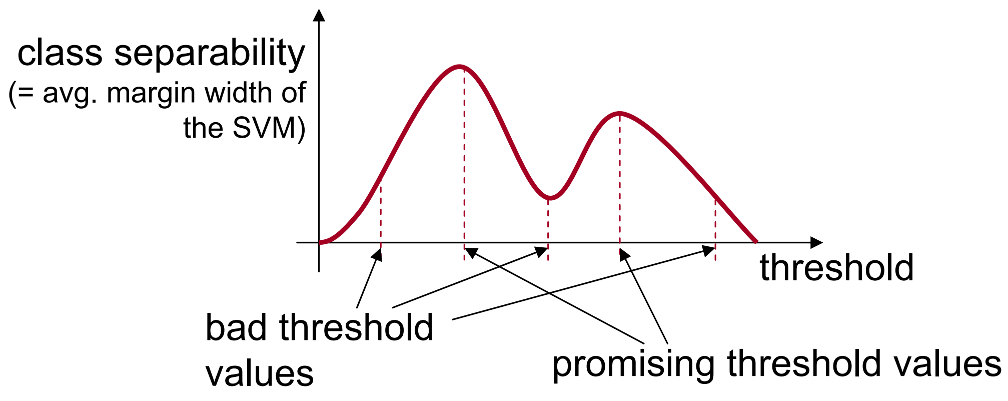


Figure 8.4: Determination of the most promising threshold values.

8.2.3 Determining the Optimal Threshold

Having determined the separation score in one of the two ways described in the previous sections for a range of interesting values, we can consider a quality curve over all these threshold values (cf. Figure 8.4). In such a separability diagram, we plot the examined threshold values along the x-axis and the corresponding separation scores along the y-axis. From this diagram we can easily determine those threshold values which are the most promising values for the clustering step of the complete dataset.

Let us note that the separability diagram not only helps to detect a certain threshold for cluster analysis, but also helps to readjust the parameter setting if the first cluster analysis returns dissatisfying results or the user wants to confirm the validity of the results by alternative threshold parameterizations.

8.3 Evaluation

In this section, we present the results of a number of experiments performed on several time series datasets.

8.3.1 Datasets and Methods

For our evaluation, we used several real-world and synthetic datasets as described in Chapter 3. In particular we used the gene expression datasets (cf. Section 3.3) GDS38 and GDS30 which will be denoted by "DS1" and "DS2" in the following. "DS3" denotes a subset of our audio dataset consisting of 36 classes with in total 756 time series. The length of the time series varies up to 30000 values per sequence. Furthermore we used the Trace dataset ("DS4"), the GunxX dataset ("DS5"), and the CBF dataset ("DS6"). While these datasets were used for a more systematical evaluation of our approach, we also evaluated the relevance of our technique on the air pollution dataset and the gene expression dataset. The gene expression dataset was also used to evaluate the relevance of different threshold levels with respect to different classification systems. We used different levels of the hierarchical classification system of the Gene Ontology [ABB$^+$00] to derive different classification systems.

In order to evaluate the quality of different thresholds, we applied the density-based clustering method OPTICS [ABKS99] for the cluster analysis step. We used OPTICS due to its robustness with respect to the data distribution and the parameter setting. Of course, any other clustering method is also applicable. We evaluated the quality of the obtained clusters by calculating the rand index [HBV01] and the average entropy. The average entropy is an unsupervised quality measure, i.e. no reference clustering is required. The entropy corresponds to the impurity of the detected clusters. Let $p_{j,i}$ be the relative frequency of the class label $label_j$ in the cluster C_i. Let $|C_i|$ be the cardinality of cluster C_i. Then the average entropy is computed as

$$avgEntropy = \frac{\sum_{C_i} |C_i| * \left(- \sum_{label_j} p_{j,i} \log(p_{j,i}) \right)}{\sum_{C_i} |C_i|}$$

Lower average entropy values correspond to a higher clustering quality. The rand index evaluates the clustering results with respect to a reference clustering. We use the classification system as a reference clustering, i.e. each class is considered a cluster. Higher values of the rand index indicate a better clustering result.

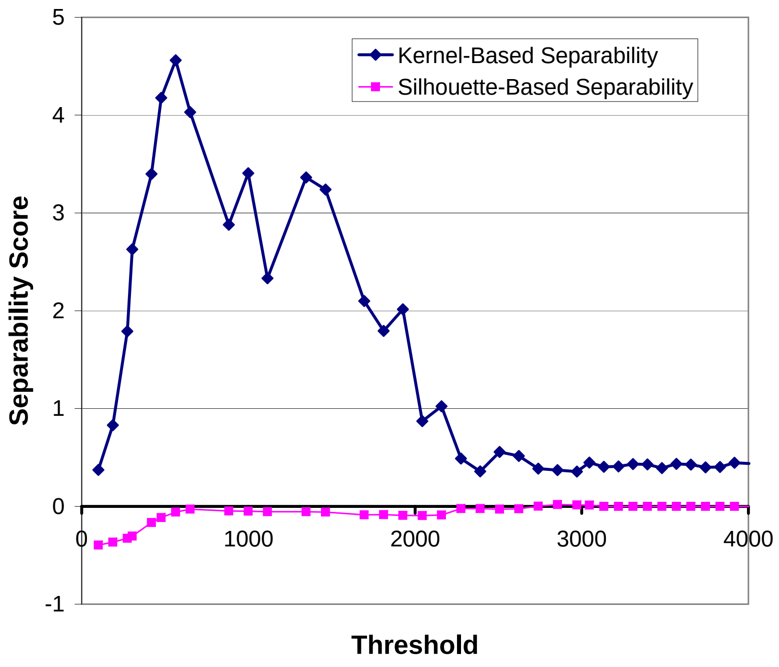


FIGURE 8.5: Separability plot on DS3.

8.3.2 Experimental Results

Validation of the Separation Score

We investigate the effectiveness of semi-supervised threshold queries which are used to find the optimal threshold value by means of a training dataset. In the first experiment we evaluated the relevance of our approach. In particular we analyzed whether choosing a promising threshold indeed leads to better clustering results than choosing an arbitrary threshold where no information about the dataset characteristics was used to select the threshold value.

At first we calculated the separability score (cf. Section 8.2) of the classes for varying threshold values. Figure 8.5 shows the results for the DS3 dataset and Figure 8.6 shows the results for the DS6 dataset. Obviously, different threshold values lead to different values for the separability of the classes. As explained in Section 8.2, high separation score values should indicate threshold values which are promising for the cluster analysis.

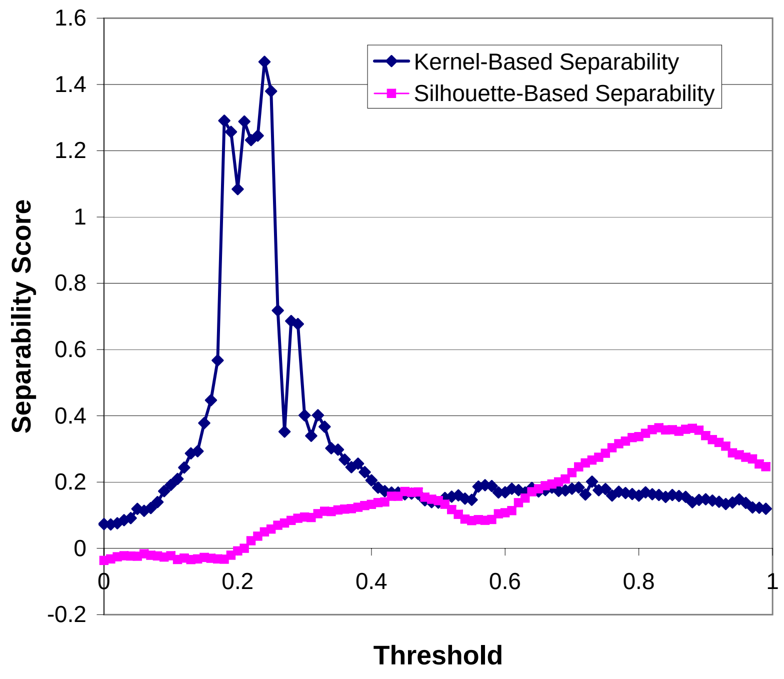


FIGURE 8.6: Separability plot on DS6.

To evaluate this, we clustered the time series for two different threshold values τ_+ and τ_- and determined the rand index and the average entropy [HBV01]. For example, the threshold value $\tau_+ = 710$ which corresponds to a high separation score on the DS3 dataset resulted in a rand index equal to 0.97. Contrary, when using a threshold value of $\tau_- = 3064$ the rand index decreased to 0.61. Similar results were observed for other levels, for other threshold values, and on other datasets. Figure 8.7 depicts the rand index and Figure 8.8 depicts the average entropy for each dataset and for a promising threshold value (τ_+) as well as for a threshold corresponding to a low separation score (τ_-). The results show that the clustering analysis based on threshold τ_+ always outperforms the quality of the clustering based on the τ_- threshold. So, our experiments confirm the general idea of our semi-supervised analysis approach.

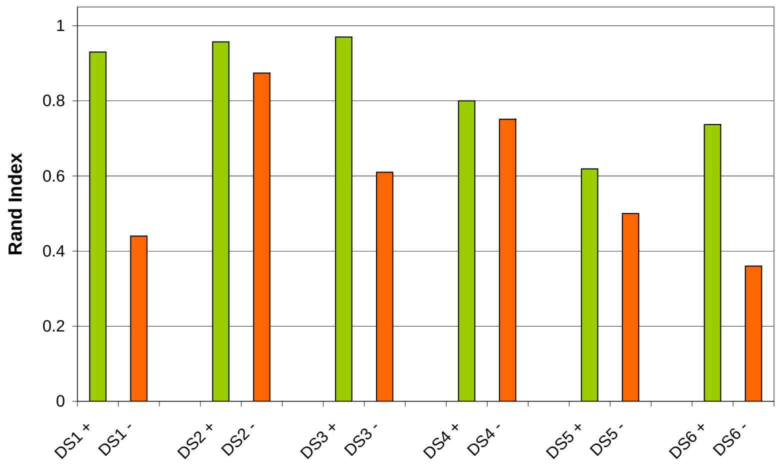


FIGURE 8.7: Rand Index for threshold values corresponding to high and low separation scores.

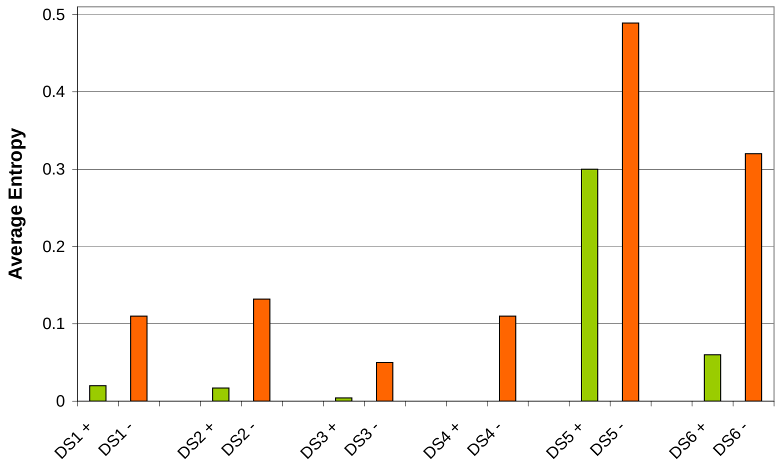


FIGURE 8.8: Average entropy for threshold values corresponding to high and low separation scores.

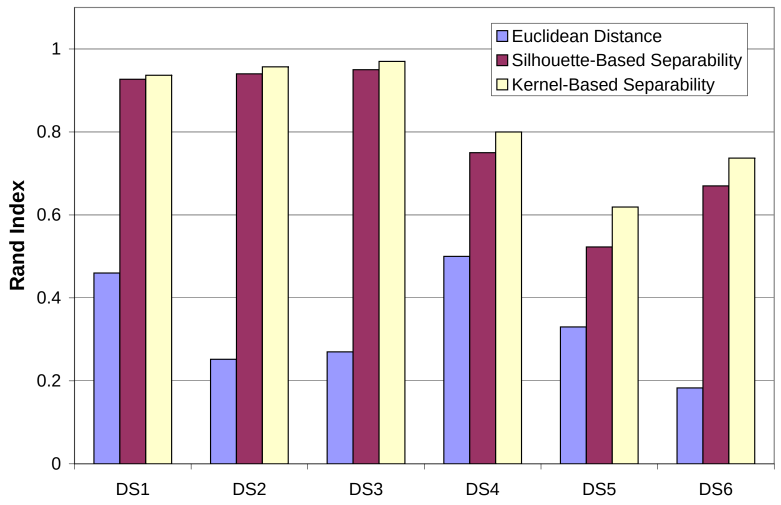


FIGURE 8.9: Comparison of clustering results.

Calculation of the Separation Score

Next, we compared the two methods introduced for the calculation of the separation score, the silhouette-based approach and the kernel-based approach as described in Section 8.2.2. For the sake of comparison, we also included clustering results obtained with the Euclidean distance. The rand index values for all 6 datasets are presented in Figure 8.9. In terms of the rand index, the kernel-based approach clearly outperforms the silhouette-based approach. Though, the silhouette-based approach yields higher quality clusterings as the Euclidean distance. The exact rand index and entropy values for all datasets are given in Table 8.1. In terms of entropy, the kernel-based approach is the best approach for 3 out of the 6 datasets, while on two datasets the silhouette-based technique yields the best clusterings. In summary, the semi-supervised analysis outperforms more straightforward similarity measures that can not be adapted to a specific dataset. Furthermore, the kernel-based separation score yields better clustering results as the separation score based on the silhouette coefficient. In the following, we restrict our experiments to the kernel-based approach.

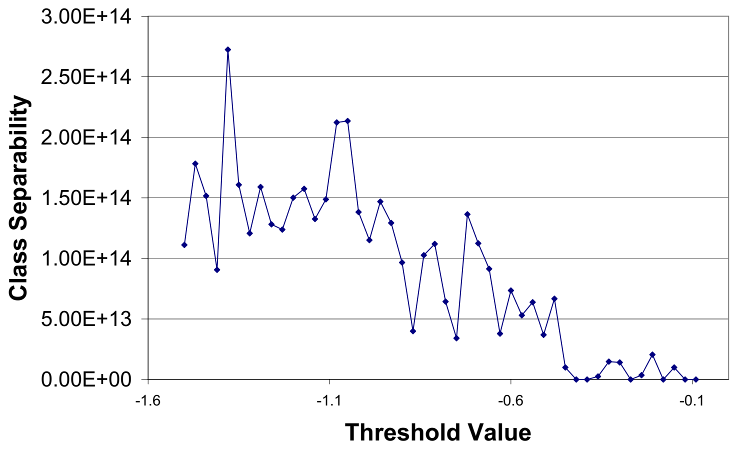
Quality Measure	Dataset	Euclidean Distance	Silhouette Coefficient	Kernel-Based
Rand Index	DS1	0.46	0.927	**0.9367**
	DS2	0.252	0.94	**0.957**
	DS3	0.27	0.95	**0.97**
	DS4	0.5	0.75	**0.8**
	DS5	0.33	0.5228	**0.619**
	DS6	0.183	0.67	**0.737**
Entropy	DS1	**0.0067**	0.047	0.026
	DS2	0.018	0.0277	**0.017**
	DS3	0.9	0.01	**0.0042**
	DS4	0.89	0.018	**0.001**
	DS5	0.8	**0.02**	0.3
	DS6	0.96	**0.025**	0.06

Table 8.1: Clustering results for different similarity measures.

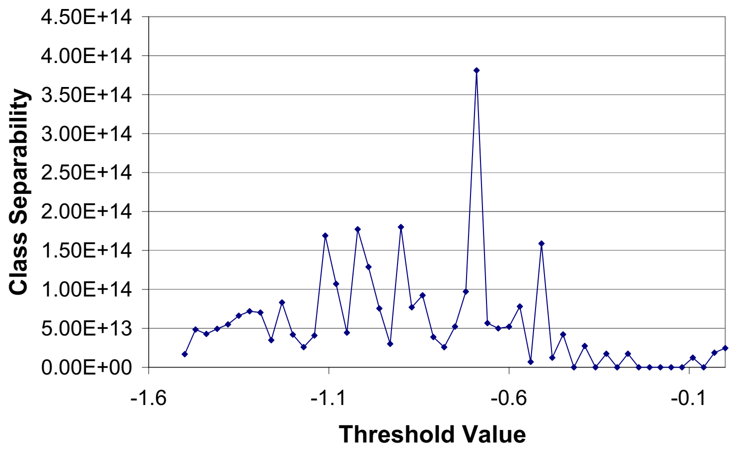
Adjustability to Different Training Classes

In the next experiment, we explored the question how the optimal threshold values change when the expected results change, i.e. when the focus of the query changes. This is a very interesting question as it reflects the possibility of different aspects in a dataset that might be relevant to a domain expert.

The following experiments were performed on the dataset DS1. For the first experiment, depicted in Figure 8.10(a) we used the GO functional classes on level 3. Afterwards we changed the focus of our analysis to the GO level 6. The results are depicted in Figure 8.10(b). Interestingly we indeed obtained different optimal threshold values for different purposes of the analysis. So, the biological classes on GO level 3 can be best distinguished using a different threshold than on GO level 6. This observation could provide a useful start-point to a biologist for further experiments and analysis.



(a) Separability curve for GO level 3.



(b) Separability curve for GO level 6.

FIGURE 8.10: Separability curves for different training classes.

Sensitivity to Incomplete Training Data

In our next experiment, we examined the sensitivity of our approach to missing classes in the training data. A low sensitivity corresponds to the ability to detect unknown knowledge. It is interesting, whether our analysis finds those classes which are existent in the training dataset, but which are not used in the training phase. The results for different datasets are depicted in Figure 8.11 in terms of the rand index and in Figure 8.12 in terms of the average entropy.

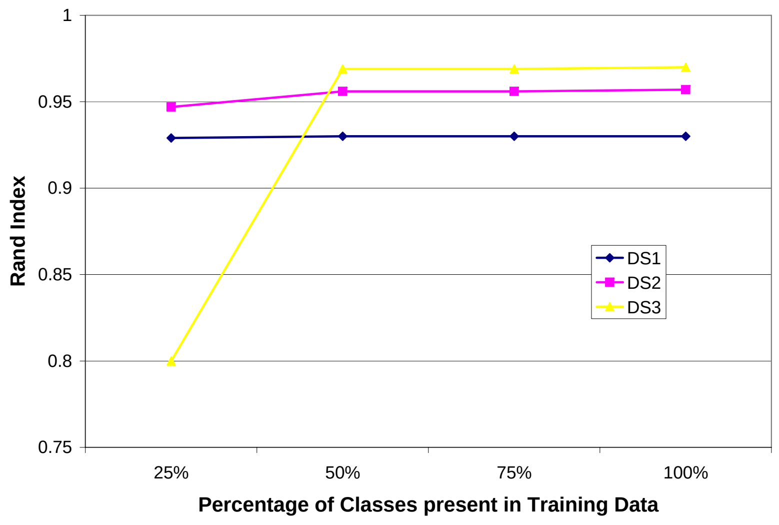


Figure 8.11: Impact of missing training data (Rand Index).

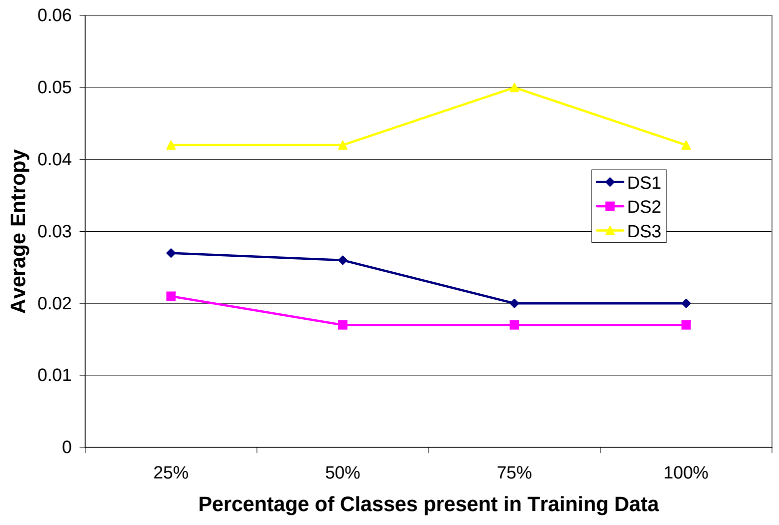


Figure 8.12: Impact of missing training data (Average Entropy).

Obviously, the completeness of the detected classes increases with an increasing number of classes used for the training phase. However, a small amount of training classes is sufficient to find nearly the complete set of classes within our dataset. This is of special interest for domain specialists, as very often only partial information is available for a dataset. Starting from this partial knowledge, new information might arise using our approach.

Analysis Results on Real-World Scientific Datasets

Finally, we evaluated the usefulness of our approach on real-world datasets. We examined time sequences of the air pollution dataset representing particulate matter parameters (M_{10}) derived from rural and urban sensor stations. The threshold-based analysis shows that the pollution with particle components in the city differs considerably from the pollution in rural regions.

The results on the gene expression dataset were also very interesting. Indeed, we retrieved functionally related genes in most of the reported clusters. For example, gene CDC25 and gene CIK3 were located in the same cluster. Both genes play an important role during the mitotic cell cycle. Furthermore, genes DOM34 and MRPL17 were in the same cluster as two genes that are not yet labeled (ORF-names: YOR182C and YGR220C, respectively). However all four genes are participating in the protein biosynthesis. In particular, our proposed analysis tool can be used to predict the function of genes whose biological role is not resolved yet.

8.4 Conclusions

In this chapter, we proposed a framework for semi-supervised cluster analysis using adaptable threshold similarity. In particular, we proposed a method to adapt the threshold by learning the optimal threshold from a small training set in order to yield an accurate clustering of the entire time series. In our experimental evaluation, we showed that our proposed approach yields valuable clustering results, even if only partial information is available for adapting the threshold to an optimal value. Beside the analysis of a dataset according to specific class labels, our approach can help to find unknown but potentially useful knowledge.

Chapter 9

T-Time: A Data Mining Tool for Time Series Data

Data mining on time series data can yield important insights. As explained in Chapter 2, a lot of similarity measures for time series have been developed. In this thesis, we introduced further similarity functions for time series. The relevance of a certain similarity measure is usually domain specific. To compare the data mining results for different similarity measures as well as the suitability of dimensionality reduction techniques, we implemented *T-Time*, a time series data mining tool.

T-Time implements a visual data mining approach that presents the data in a clear and user-friendly way in order to enable interactive data exploration, e.g. cluster analysis. In particular, *T-Time*

- assists the user in identifying potentially interesting threshold values;
- enables the visual and interactive exploration of other data analysis parameters;
- allows the user to interactively and visually extract novel knowledge from a large amount of data derived from data mining algorithms.

The main focus of *T-Time* therefore is the interactive and visual analysis of the impact of different threshold values on the results of data mining tasks. The concept of our application supports the extraction of novel insights in supervised as well as in unsupervised settings. If class labels are available, the user can easily scan for threshold values that yield high classification accuracies in cross-validation experiments. This subsequently allows for the identification of ranges of important amplitudes of the time series, i.e. ranges where small differences in the absolute values account for large differences in the meaning (different classes) of the time series. But even in an unsupervised situation where no pre-classified time series are available, *T-Time* can be very helpful. By a quick visual inspection of several clustering results derived for example by OPTICS [ABKS99] it is possible to discover important and interesting thresholds based on their ability to form distinct cluster structures.

9.1 System Overview

T-Time is a Java 1.5. application with a graphical user-interface. A number of distance measures, dimensionality reduction techniques, and data mining approaches have been included. The architecture of *T-Time* allows for an easy integration of further components. In the following we briefly describe the application of *T-Time* for unsupervised and supervised situations.

9.1.1 Visual Comparison of Time Series

The main control window of *T-Time* allows the user to import collections of time series. Figure 9.1 depicts the corresponding view for an imported dataset. The left area of the dataset window features a textual entry for each time series. If available, class labels appear in brackets.

To the right, the time series are displayed as diagrams for a visual inspection. Time series of different classes are displayed in different colors. By selecting several time series simultaneously, it is possible to directly compare them. In Figure 9.1, two time series belonging to different classes have been

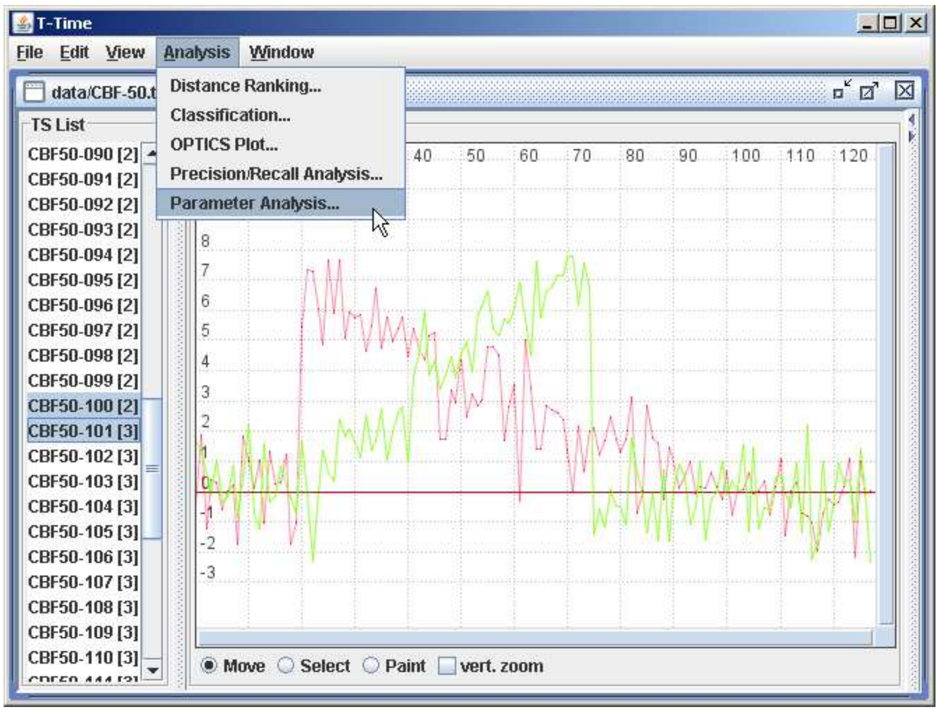
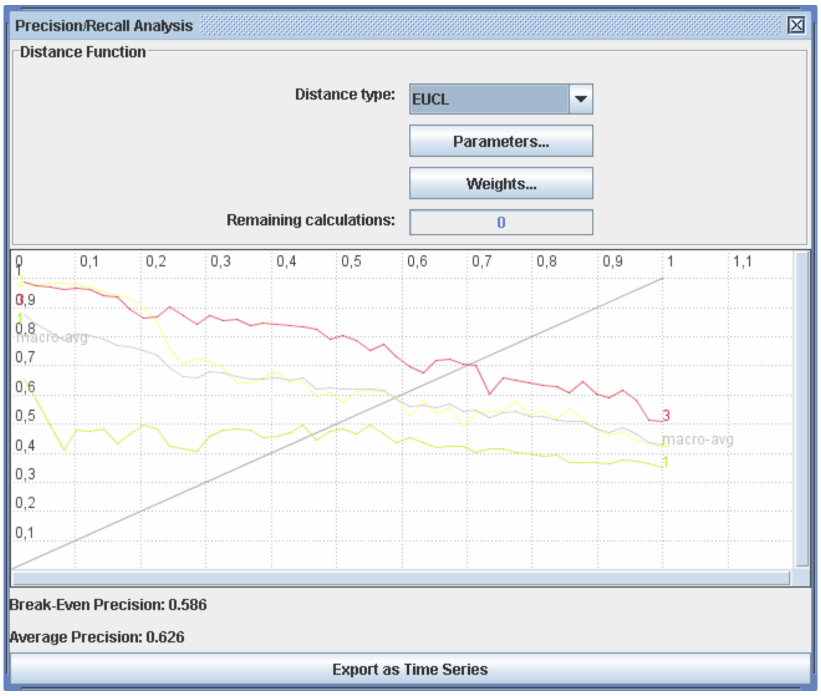


Figure 9.1: Main Window of *T-Time*.

selected. After selecting all or a subset of the time series, the user can start one of the numerous data mining algorithms included in the tool. The following sections show how different threshold values influence unsupervised as well as supervised data mining tasks.

9.1.2 Supervised Analysis

If pre-classified time series are available, it is possible to perform a number of different analysis tasks using several distance measures. We included a kNN classifier in *T-Time*. Different similarity measures can be compared by means of cross-validations experiments. Furthermore, *T-Time* can calculate precision/recall plots for the different classes of a certain dataset. An example plot is depicted in Figure 9.2. The results for different classes are presented in different colors.



Distance type: EUCL

Remaining calculations: 0

Break-Even Precision: 0.586

Average Precision: 0.626

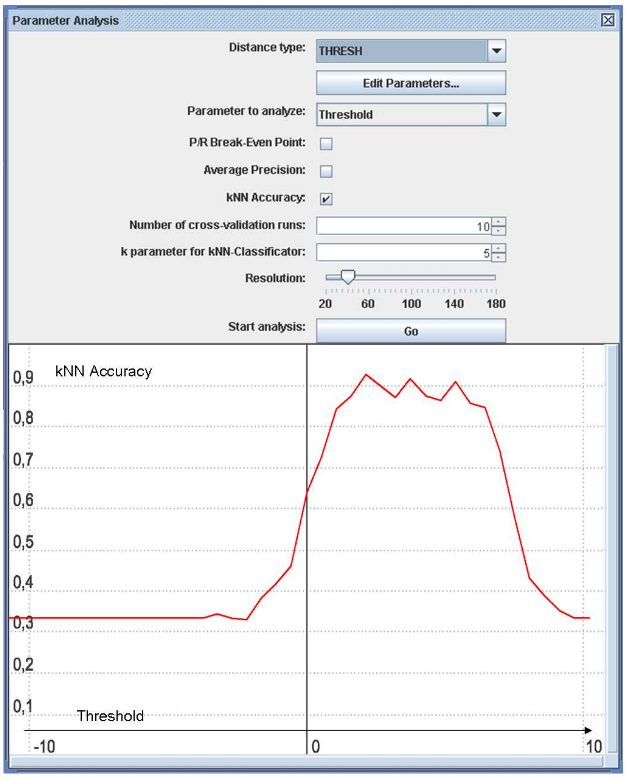
FIGURE 9.2: Precision/recall analysis.

9.1.3 Unsupervised Analysis

Even if only unlabeled time series objects are available, *T-Time* can be of great help to analyze the impact of different distance functions and especially to identify ranges of distinguishing threshold values. While in principle every clustering approach could be used, we decided to integrate OPTICS [ABKS99] into *T-Time* as its results can easily be interpreted visually. OPTICS is a variant of single-link clustering that avoids the single-link effect by using a density estimator for data grouping. OPTICS provides a linear ordering of the data objects that can be visualized by means of a so-called reachability diagram. This visualization of the hierarchical clustering structure is much clearer compared to dendrograms. Valleys in this reachability diagram indicate clusters. Of course, any other clustering or visualization technique can be modularly integrated in the analysis process.

9.2 Threshold-Based Data Mining

One of the most useful *T-Time* applications is the automatic identification of distinguishing threshold values for threshold-based distance functions. In Figure 9.3, an example output is depicted. For a number of threshold values along the x-axis, classification accuracy values are plotted in y-axis direction. Usually one or a few distinct ranges of suitable threshold values can be identified in this way. In the depicted example, the most distinguishing threshold values can be found in the range between 3 and 6. We observed such a distinct range of meaningful threshold values for most real-world datasets. Based on such kind of information and depending on the application domain, conclusions about critical time series values can be drawn.


Parameter Analysis

Distance type: THRESH

Edit Parameters...

Parameter to analyze: Threshold

P/R Break-Even Point: ☐

Average Precision: ☐

kNN Accuracy: ✓

Number of cross-validation runs: 10

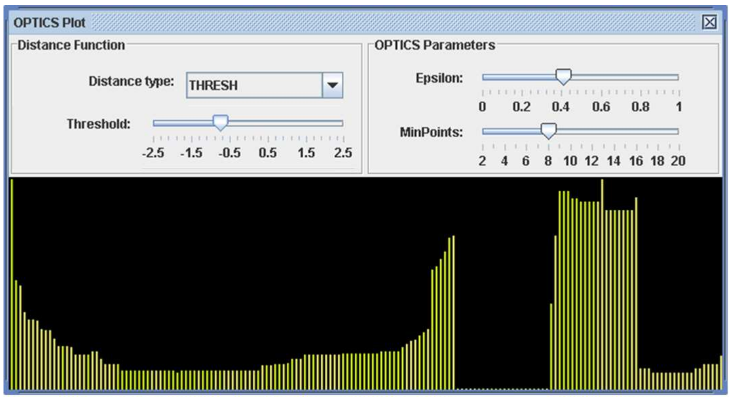
k parameter for kNN-Classificator: 5

Resolution: 20 60 100 140 180

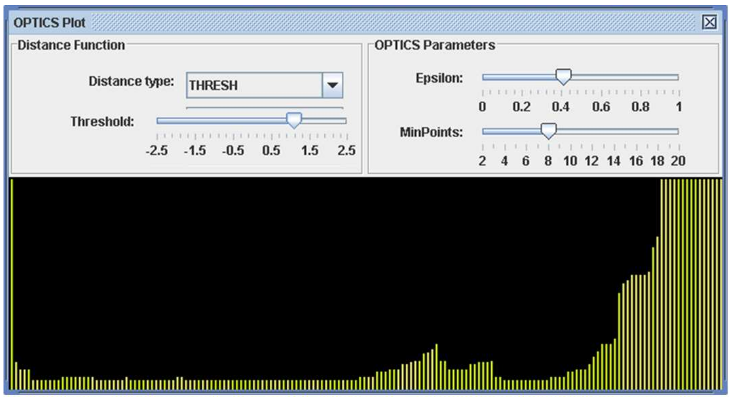
Start analysis: Go

Figure 9.3: Identification of distinguishing thresholds.

If no class labels are available, the impact of different threshold values can be evaluated by means of the corresponding cluster structure. In Figure 9.4 clustering results are depicted for different threshold values. In the depicted example, the threshold $\tau_1 = -0.75$ results in 3 clearly separated OPTICS clusters while the threshold $\tau_2 = 1.1$ yields only one large cluster. So, τ_1 could be more interesting for the user than threshold $\tau_2 = 1.1$, especially if for example the number of clusters corresponds to the number of subtypes of a certain disease.


OPTICS Plot
Distance Function
Distance type: THRESH
Threshold:
OPTICS Parameters
Epsilon:
MinPoints:

(a) Threshold $\tau_1 = -0.75$


OPTICS Plot
Distance Function
Distance type: THRESH
Threshold:
OPTICS Parameters
Epsilon:
MinPoints:

(b) Threshold $\tau_2 = 1.1$

FIGURE 9.4: Unsupervised threshold analysis.

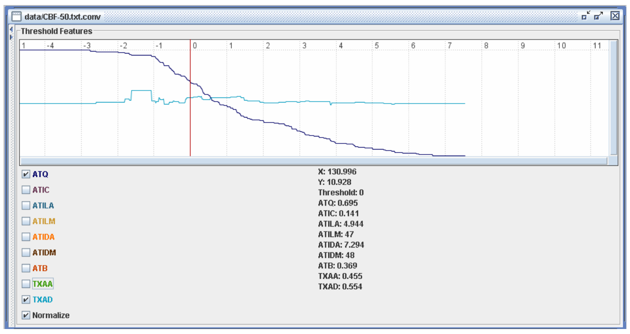
We applied *T-Time* to a set of classified time series representing human gene expression data. We used a dataset of the Gene Expression Omnibus [BTW+06] containing gene expression profiles of proliferating normal peripheral blood mononuclear cells (PBMC) infected with HIV type 1 RF assessed at five postinfection time points compared with those of matched uninfected PBMC. We then tried to detect pathological genes. The idea is to derive quality curves as depicted in Figure 9.3 for each subset of the dataset corresponding to a certain gene. As expected we found that most genes yielded no distinct peak when computing the quality curves with respect to the classification system (healthy vs. infected cells). However, a few genes did yield such a distinguished region. That means these genes act significantly different in healthy and in infected cells and are thus candidates to be highly pathological. For example, one of these genes is *NFYC* which plays a role in the transcription of the *MHCII* genes that are blocked by an HIV protein. Another gene featuring a noticeable quality curve is *PLAUR* whose expression is known to be affected by an HIV infection [SOBM02].

Furthermore we successfully applied *T-Time* to a dataset that consists of gene expression data corresponding to patient responses to the drug 'Tamoxifen'. The dataset was taken from the Pharmacogenetics and Pharmacogenomics Knowledge Base [KCC+01]. We observed a dramatically changing cluster structure when varying the threshold. In case of $\tau = 0$ we observed 3 clusters. When changing τ to -0.3, we can only observe 2 clusters with a completely different cluster membership of patients. Thus, with different thresholds, we can cluster the patients according to varying phenotypes. A domain expert could use this information to identify important genes and crucial expression levels.

9.3 Amplitude-Level-Based Data Mining

T-Time is also able to display various Amplitude-level features as described in Chapter 5. Two sample ALFs are depicted in Figure 9.5.

Again, in a supervised setting, it is possible to automatically analyze the impact of different relevant thresholds on the classification quality. Like for the threshold-based analysis, this can yield new insights about relevant amplitude values. In Figure 9.6 an example output of a parameter analysis

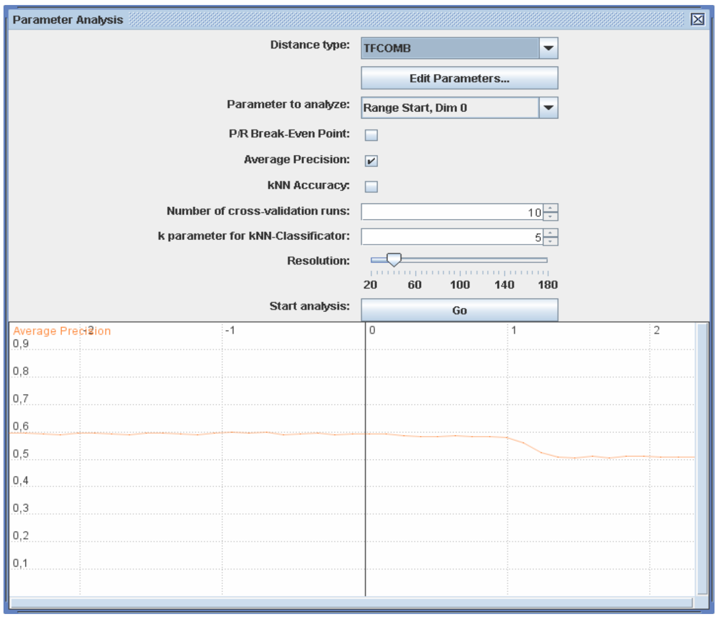


- [✓] ATQ
- [ ] ATIC
- [ ] ATILA
- [ ] ATILM
- [ ] ATIDA
- [ ] ATIDM
- [ ] ATB
- [ ] TXAA
- [✓] TXAD
- [✓] Normalize

X: 130.996
Y: 10.928
Threshold: 0
ATQ: 0.695
ATIC: 0.141
ATILA: 4.944
ATILM: 47
ATIDA: 7.294
ATIDM: 48
ATB: 0.369
TXAA: 0.455
TXAD: 0.554

Figure 9.5: Amplitude-level features in T-Time.

run is depicted. The maximal relevant threshold was kept constant, while we varied the minimal threshold. Each setting actually corresponds to a slightly different similarity measure. For each of these amplitude-based similarity measures, the average classification accuracy is reported. 9.5.

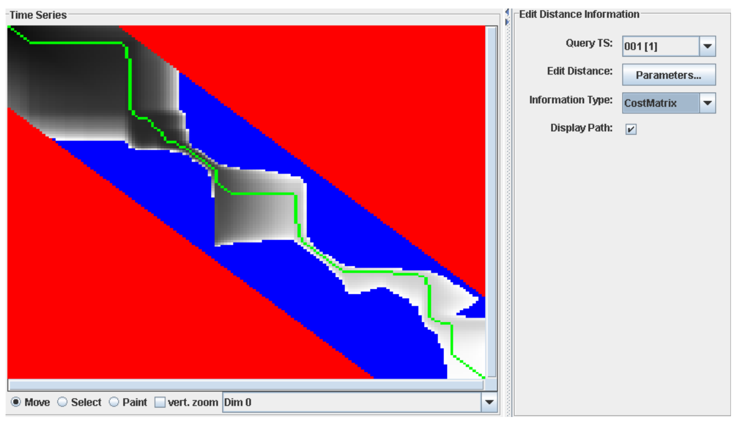


Distance type: TFCOMB
Parameter to analyze: Range Start, Dim 0
P/R Break-Even Point: [ ]
Average Precision: [✓]
kNN Accuracy: [ ]
Number of cross-validation runs: 10
k parameter for kNN-Classificator: 5
Resolution:
Start analysis:

Figure 9.6: Analysis of different relevant amplitude ranges.

9.4 Further Applications

T-Time can furthermore be used to compare traditional similarity measures and dimensionality reduction techniques with each other. We implemented several Minkowski distances, several variants of the DTW, as well as the dimensionality reduction techniques mentioned in Section 2.5.3. For the DTW-based distance measures it is possible to display the cost matrix and the optimal DTW path (see Figure 9.7).



Query TS: 001 [1]
Edit Distance: Parameters...
Information Type: CostMatrix
Display Path: ✓
(● Move ○ Select ○ Paint) ☐ vert. zoom — Dim 0

FIGURE 9.7: Cost matrix and optimal path of dynamic time warping.

9.5 Conclusions

In this chapter we presented *T-Time*, a data mining tool designed for the comparison of several distance measures, especially threshold-based distance measures. A main advantage of *T-Time* is its ability to support visual data mining, especially when it comes to identifying crucial threshold ranges. These ranges are of practical importance, as based on such observations, domain experts can gain novel insights on a given dataset. *T-Time* supports supervised as well as unsupervised analysis tasks and offers the possibility to compare new distance measures for time series to traditional approaches.

Part IV

Conclusions and Outlook

Chapter 10

Summary and Outlook

In this chapter we briefly summarize our contributions in Section 10.1. In Section 10.2 we outline several ideas for further research based on the work presented in this thesis.

10.1 Summary

In **Part II** we introduced several new similarity measures for time series capturing special notions of time series similarity.

First (see Chapter 4), we defined the new similarity measure based on thresholds. Instead of the exact comparison of two time series this similarity measure rather considers threshold-exceeding events. We presented an index structure which efficiently supports queries for the threshold-based similarity measure. This index structure allows for the specification of the threshold value at query time. We showed how time series can be converted into a suitable representation and be stored in such a way, that queries using an arbitrary threshold value can be supported. In an experimental section we showed how the proposed index structure saved computational cost. On real-world datasets we proved the practical relevance of the threshold-based similarity measure.

Next (see Chapter 5), we introduced the amplitude-level-based similarity. This similarity measure considers a relevant amplitude range rather than a single threshold. In contrast to traditional approaches which aggregate global feature values along the time dimension, we capture local characteristics and monitor their change for different amplitude values.

In Chapter 6 we introduced the interval-focused similarity measure, where a user can specify one or several time intervals that should be considered for the calculation of the similarity value. We showed how to use an existing index structure, the RI-tree, in order to store interval-based representations of time series. We outlined how these interval boxes can be used to calculate a filter distance in order to prune true drops, or to include true hits without accessing the complete time series.

In the last chapter (see Chapter 7) of this part, we introduce the novel time series type of uncertain time series. We formalized our approach and showed how a huge part of the actually required calculations can be saved using the proposed approximation for uncertain time series.

In **Part III** we showed how the newly defined similarity measures can be used in data mining tasks.

In Chapter 8 we focused on an important question for the threshold-based similarity, the choice of a suitable threshold value. We showed how partial knowledge can be used to get a first idea about promising thresholds in order to extract novel knowledge from a dataset.

The last chapter of this part (see Chapter 9) presented a data mining tool for time series. We gave an overview of the supported data mining tasks and the included distance measures.

In short, our main contributions in this thesis are the following:

- We introduce 4 new similarity measures for time series.
- For each similarity measure we propose algorithms and index structures that allow for an efficient calculations of queries.
- In the experimental sections we presented the results of experiments

comparing the efficiency of our approaches compared to competing approaches.

- We furthermore gave examples for the relevance of the new similarity measures.
- We showed how new similarity measures can model alternative concepts of similarity and applied these new measures to data mining tasks.

10.2 Outlook

In this chapter, we describe potential directions for future work.

10.2.1 Threshold-Based Similarity

It might be of interest to develop a more approximating version of our index structure. It may be possible to group threshold-crossing time intervals together while not introducing too much of an error. Connected segments that do not differ too much in their direction could be summarized by cylinder-like structures. A query using these structures could provide a first filter distance which would be more efficient to compute.

10.2.2 Amplitude-Level-Based Similarity

We observed that, different reduction techniques are more or less suitable for different feature sequences, and sometimes the best representation is the uncompressed one. This observation suggests a potential improvement of our method. Instead of using the same compression techniques for all feature sequences, it may be beneficial to develop a method which automatically chooses the most suitable compression technique for a given dataset and for each ALF feature sequence.

A second starting point is the development of further amplitude-level features. With the basic ALFs presented in this work, already high quality results could be observed. So the method may benefit from more complex ALFs.

10.2.3 Interval-Focused Similarity

So far, we have shown how to efficiently support interval-focused similarity queries for Minkowski norms. We plan to extend our idea to the DTW distance.

Furthermore, the strategy for the interval box generation assumes certain data distributions. While we have explained that these assumptions are sensible ones, the method might nonetheless benefit from the possibility to model a certain distribution, in case the distribution of amplitude values of the query time series is known in advance.

10.2.4 Similarity for Uncertain Time Series

For the uncertain time series we plan to investigate potential solutions for the case of correlated time series as well.

So far, we have introduced probabilistic queries where the distance parameter ε was constant. It is an interesting question how to define kNN queries, as it is not obvious which of the two possibilities is the next neighbor to a query: the time series that is most probably relatively near to the query, or the time series that is very near to a time series but with a lower probability.

List of Figures

List of Tables

Bibliography

[ABB$^+$00] M. Ashburner, C.A. Ball, J.A. Blake, D. Botstein, H. Butler, J.M. Cherry, A.P. Davis, K. Dolinski, S.S. Dwight, J.T. Eppig, M.A. Harris, D.P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J.C. Matese, J.E. Richardson, M. Ringwald, G.M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[ABKS99] M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the SIGMOD Conference, Philadelphia, PA*, 1999.

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. "Efficient Similarity Search in Sequence Databases". In *Proc. 4th Conf. on Foundations of Data Organization and Algorithms*, 1993.

[AKK$^+$06a] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Semi-supervised threshold queries on pharmacogenomics time sequences. In *Proceedings of the 4th Asia Pacific Bioinformatics Conference (APBC), Taipei, Taiwan*, 2006.

[AKK$^+$06b] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Similarity search on time series based on threshold queries. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT), Munich, Germany*, 2006.

[AKK$^+$06c] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Threshold similarity queries in large time series databases. In *Proceedings of the 22st International Conference on Data Engineering (ICDE), Atlanta, GA*, 2006.

[AKK$^+$06d] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Time series analysis using the concept of adaptable threshold similarity. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM), Vienna, Austria*, 2006.

[AKK$^+$06e] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Tquest:threshold query execution for large sets of time series. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT), Munich, Germany*, 2006.

[AKK$^+$07] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Interval-focused similarity search in time series databases. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA), Bangkok, Thailand*, 2007.

[AKK$^+$08a] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Similarity search in multimedia time series data unsing amplitude-level features. In *Proceedings of the 14th IEEE International MultiMedia Modelling Conference (MMM), Kyoto, Japan*, 2008.

[AKK$^+$08b] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. T-time: Threshold-based data mining on time series. In *Proceedings of the 24th International Conference on Data Engineering (ICDE), Cancun, Mexico*, 2008.

[AM99] R. J. Alcock and Y. Manolopoulos. Time-series similarity queries employing a feature-based approach. In *Proceedings of the 7th Hellenic Conference on Informatics, Ioannina, Greece*, 1999.

[BC94] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Proc. AAAI-94 W. on Knowledge Discovery and Databases.*, pages 229–248, 1994.

[Bel61] R. Bellman. "Adaptive Control Processes: A Guided Tour". In *Princeton University Press*, 1961.

[Ber02] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[BKS$^+$04] Benjamin Bustos, Daniel Keim, Dietmar Saupe, Tobias Schreck, and Dejan Vranic. Automatic selection and combination of descriptors for effective 3d similarity search. In *Proceedings of the IEEE 6th International Symposium on Multimedia Software Engineering, Washington, D.C.*, 2004.

[BKSS90] N. Beckmann, H.-P. Kriegel, B. Seeger, and R. Schneider. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the SIGMOD Conference, Atlantic City, NJ*, 1990.

[BM03] M. Bilenko and R.J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, D.C.*, 2003.

[BPS06a] C. Böhm, A. Pryakhin, and M. Schubert. The gauss-tree: Efficient object identification of probabilistic feature vectors. In *Proceedings of the 22st International Conference on Data Engineering (ICDE), Atlanta, GA*, 2006.

[BPS06b] C. Böhm, A. Pryakhin, and M. Schubert. Probabilistic ranking queries on gaussians. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management (SSDBM), Vienna, Austria*, 2006.

[BSM04] M. Bilenko, B. Sugato, and R.J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the 21st International Conference on Machine learning (ICML), Banff, Canada*, 2004.

[BTW+06] T. Barrett, D.B. Troup, S.E. Wilhite, P. Ledoux, D. Rudnev, C. Evangelista, I.F. Kim, A. Soboleva, M. Tomashevsky, and R. Edgar. Incbi geo: mining tens of millions of expression profiles–database and tools update. In *Nucleic Acids Research*, 2006.

[CBW+97] J.M. Cherry, C. Ball, S. Weng, G. Juvik, R. Schmidt, C. Adler, B. Dunn, S. Dwight, L. Riles, R.K. Mortimer, and D. Botstein. Genetic and physical maps of saccharomyces cerevisiae. *Nature*, 387:67–73, 1997. `www.yeastgenome.org`.

[CF99] K. Chan and W. Fu. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering (ICDE), Sydney, Australia*, 1999.

[CH67] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[CKP03] R. Cheng, D.V. Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of the SIGMOD Conference, San Diego, CA*, 2003.

[CKP07] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluation of probabilistic queries over imprecise data in constantly-evolving environments. *Information Systems*, 32(1):104–130, 2007.

[CM99] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *Proceedings of the 15th International Conference on Data Engineering (ICDE), Sydney, Australia*, 1999.

[CN04] Y. Cai and R. Ng. Index spatio-temporal trajectories with chebyshev polynomials. In *Proceedings of the SIGMOD Conference, Paris, France*, 2004.

[CXP$^+$04] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J.S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada*, 2004.

[DBE99] A. Demiriz, K. P. Bennett, and M. J. Embrechts. Semi-supervised clustering using genetic algorithms. In *Intelligent Engineering Systems Through Artificial Neural Networks 9*, pages 809–814, 1999.

[DGM97] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. In *Proceedings of the 1st European Conference on Principles of Knowledge Discovery and Data Mining (PKDD), Trondheim, Norway*, pages 88–100, 1997.

[DGM$^+$04] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada*, 2004.

[DGM05] A. Deshpande, C. Guestrin, and S. R. Madden. Using probabilistic models for data management in acquisitional environments. In *Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA*, 2005.

[DMN97] K. Deng, A. Moore, and M. Nechyba. Learning to recognize time series: Combining arma models with memory-based learning. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation, Washington, D.C.*, 1997.

[DP97] P. Domingos and M.J Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[EDL02] R. Edgar, M. Domrachev, and A.E. Lash. Gene expression omnibus: Ncbi gene expression and hybridization array data repository. *Nucleic Acids Research*, 30(1):207–210, 2002.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In *Proceedings of the 2nd ACM International Conference on Knowledge Discovery and Data Mining (KDD), Portland, OR*, 1996.

[EM97] T. Eiter and H. Mannila. Distance measure for point sets and their computation. In *Acta Informatica, 34*, pages 103–133, 1997.

[EZZ04] C.h F. Eick, N. M. Zeidat, and Z. Zhao. Supervised clustering - algorithms and benefits. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Boca Raton, FL*, pages 774–776, 2004.

[FGB02] A. Faradjian, J. Gehrke, and P. Bonnet. GADT: A probability space ADT for representing and querying the physical world. In *Proceedings of the 18th International Conference on Data Engineering (ICDE), San Jose, CA*, 2002.

[FL95] C. Faloutsos and K.-I. Lin. "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets". In *Proceedings of the SIGMOD Conference, San Jose, CA*, pages 163–174, 1995.

[FM84] A. Fournier and D. Y. Moniwno. Triangulating simple polygons and equivalent problems. *ACM Transactions on Graphics*, 3(2):153–174, 1984.

[FPSS96] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *Proceedings of the 2nd ACM International Conference on Knowledge Discovery and Data Mining (KDD), Portland, OR*, 1996.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Maolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the SIGMOD Conference, Minneapolis, MN*, 1994.

[GFKS02] T. Gärtner, P. A. Flach, A. Kowalczyk, and A. J. Smola. Multi-instance kernels. In *Proceedings of the 19th International Conference on Machine Learning (ICML), Sydney, Australia*, 2002.

[GG84] V. Gaede and O. Günther. Multidimensional access methods. *Computing Surveys*, 30(2), 1984.

[GRS98] S. Guha, R. Rastogi, and K. Shim. "CURE: an efficient clustering algorithm for large databases". In *Proceedings of the SIGMOD Conference, Seattle, WA*, 1998.

[GS00] X. Ge and P. Smyth. Deformable markov model templates for time-series pattern matching. In *Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Boston, MA*, pages 81–90, 2000.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the SIGMOD Conference, Boston, MA*, 1984.

[HBV01] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. "On Clustering Validation Techniques". In *Intelligent Information Systems Journal*, 2001.

[HK98] A. Hinneburg and D.A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the 4th ACM International Conference on Knowledge Discovery and Data Mining (KDD), New York City, NY*, 1998.

[HK01] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Academic Press, 2001.

[HS95] G. Hjaltason and H. Samet. Ranking in spatial databases. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases, Portland, OR*, 1995.

[KCC$^+$01] T.E. Klein, J.T. Chang, M.K. Cho, K.L. Easton, R. Fergerson, M. Hewett, Z. Lin, Y. Liu, S. Liu, D.E. Oliver, D.L. Rubin, F. Shafa, J.M. Stuart, and R.B. Altman. Integrating genotype and phenotype information: An overview of the pharmgkb project. In *The Pharmacogenomics Journal*, 2001.

[KCMP01] E. Keogh, K. Chakrabati, S. Mehrotra, and M. Pazzani. "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases". In *Proceedings of the SIGMOD Conference, Santa Barbara, CA*, 2001.

[KCPM00] E. Keogh, K. Chakrabarti, M.J. Pazzani, and S. Mehrotra. "dimensionality reduction for fast similarity search in large time series databases". *"Knowledge and Information Systems"*, 3(3):263–286, 2000.

[KF02] E. Keogh and T. Folias. The ucr time series data mining archive, 2002. `http://www.cs.ucr.edu/~eamonn/TSDMA/index.html`.

[KJF97] F. Korn, H. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proceedings of the SIGMOD Conference, Tucson, AZ*, 1997.

[KKM02] D. Klein, S.D. Kamvar, and C.D. Manning. From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *Proceedings of the 19th International Conference on Machine Learning (ICML), Sydney, Australia*, 2002.

[KKM07] H.-P. Kriegel, P. Kunath, and Renz M. Probabilistic nearest-neighbor query on uncertain objects. In *Proceedings of the 12th*

International Conference on Database Systems for Advanced Applications (DASFAA), Bangkok, Thailand, 2007.

[KKPR06] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Probabilistic similarity join on uncertain data. In *Proceedings of the 11th International Conference on Database Systems for Advanced Applications (DASFAA), Singapore, Singapore*, 2006.

[KLR04] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Seattle, WA*, 2004.

[KM04] H.-P. Kriegel and Schubert M. Classification of websites as sets of feature vectors. In *Proceedings of the International Conference on Databases and Applications (DBA), Innsbruck, Austria*, 2004.

[Kot07] S.B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.

[KP01] E. J. Keogh and M. J. Pazzani. Derivative dynamic time warping. In *Proceedings of the 1st SIAM International Conference on Data Mining (SDM), Chicago, IL*, 2001.

[KPS01a] H.-P. Kriegel, M. Pötke, and T. Seidl. "Interval Sequences: An Object-Relational Approach to Manage Spatial Data". In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD), Redondo Beach, CA*, 2001.

[KPS01b] H.-P. Kriegel, M. Pötke, and T. Seidl. Object-relational indexing for general interval relationships. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD), Redondo Beach, CA*, 2001.

[KPS05] H.-P. Kriegel, A. Pryakhin, and M. Schubert. Multi-represented knn-classification for large class sets. In *Proceedings of the 10th*

International Conference on Database Systems for Advanced Applications (DASFAA), Beijing, China, 2005.

[KR90] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: an introduction to cluster analysis.* Wiley, 1990.

[KR02] E. Keogh and C.A. Ratanamahatana. Exact indexing of dynamic time warping, 2002.

[KSF+96] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. "Fast Nearest Neighbor Search in Medical Image Databases". In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB), Bombay, India*, pages 215–226, 1996.

[LFU] *Bayerisches Landesamt für Umwelt.* `www.bayern.de/lfu`.

[LLRS97] L.V.S. Lakshmanan, N. Leone, R.B. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Transactions on Database Systems (TODS)*, 22(3):419–469, 1997.

[LS07] V. Ljosa and A. K. Singh. APLA: Indexing arbitrary probability distributions. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey*, 2007.

[LW01] M. Liu and C. Wan. Feature selection for automatic classification of musical instrument sounds. In *Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries,Roanoke, VA*, 2001.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[MZB06] D. Mitrovic, M. Zeppelzauer, and C. Breiteneder. Discrimination and retrieval of animal sounds. In *Proc. MMM*, 2006.

[NAM01] A. Nanopoulos, R. Alcock, and Y. Manolopoulos. Feature-based classification of time-series data. *Information processing and technology*, pages 49–61, 2001.

[NH94] R. Ng and J. Han. "Efficient and Effective Clustering Methods for Spatial Data Mining". In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), Santiago de Chile, Chile*, pages 144–155, 1994.

[Pam04] E. Pampalk. A matlab toolbox to compute music similarity from audio. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR), Barcelona, Spain*, 2004.

[Qui93] J.R. Quinlan. *C4.5: Programs for machine learning.* Morgan Kaufmann, San Francisco, 1993.

[RK05] C.A. Ratanamahatana and E. Keogh. Three myths about dynamic time warping data mining. In *SDM05*, 2005.

[RKBL05] C.A. Ratanamahatana, E. Keogh, A.J. Bagnall, and S. Lonardi. A novel bit level time series representation with implication for similarity search and clustering. In *Proc. PAKDD*, 2005.

[Rov02] D. Roverso. Plant diagnostics by transient classification: The aladdin approach. In *IJIS,Special Issue on Intelligent Systems for Plant Surveillance and Diagnostics*, volume 17, pages 767–790, 2002.

[Sai94] N. Saito. *Local feature extraction and its application using a library of bases.* PhD thesis, Yale University, New Haven, Connecticut, 1994.

[SBM04] B. Sugato, M. Bilenko, and R.J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Seattle, WA*, pages 59–68, 2004.

[SBS99] B. Schölkopf, C. J. C. Burges, and A. J. Smola. *Advances in Kernel Methods - -Support Vector Learning.* MIT Press, 1999.

[SCSS05] A. Schliep, I. G. Costa, C. Steinhoff, and A. Schonhuth. Analyzing gene expression time-courses. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 2(3):179–193, 2005.

[SHP98] T. Seidl and Kriegel H.-P. "Optimal Multi-Step k-Nearest Neighbor Search". In *Proceedings of the SIGMOD Conference, Seattle, WA*, 1998.

[Sib73] R. Sibson. "SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method". In *The Computer Journal*, volume 16(1), pages 30–34, 1973.

[SMKF04] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. The Princeton Shape Benchmark. In *Proceedings of the Shape Modeling International, Genova, Italy*, 2004.

[SOBM02] M. Storgaard, N. Obel, F. T. Black, and B.K. Moller. Decreased urokinase receptor expression on granulocytes in hiv-infected patients. In *Scandinavian Journal of Immunology*, 2002.

[SSZ$^+$98] P. Spellman, G. Sherlock, M. Zhang, V. Iyer, K. Anders, M. Eisen, P Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Molecular Biolology of the Cell*, 9:3273–3297, 1998.

[Sun02] X. Sun. Pitch determination and voice quality analysis using subharmonic-to-harmonic ratio. In *Proceedings. of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Orlando, Florida*, 2002.

[TCX$^+$05] Y. Tao, R. Cheng, X. Xiao, W.K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proceedings of the 31st*

International Conference on Very Large Data Bases (VLDB), Trondheim, Norway, 2005.

[Tre82] T. Tremain. The government standard linear predictive coding algorithm: Lpc-10., 1982.

[Vap95] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

[VHGK03] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, D.C.*, pages 216–225, 2003.

[WCRS01] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the 18th International Conference on Machine Learning (ICML), Williams College, MA*, pages 577–584, 2001.

[WFS04] S. Wichert, K. Fokianos, and K. Strimmer. Identifying periodically expressed transcripts in microarray time series data. *Bioinformatics*, 20(1):5–20, 2004.

[WSH06] X. Wang, K. Smith, and R. Hyndman. Characteristic-based clustering for time series data. *Data Min. Knowl. Discov.*, 13(3):335–364, 2006.

[YF00] B. K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB), Cairo, Egypt*, 2000.

[Zho05] Shi Zhong. Semi-supervised sequence classification with HMMs. *IJPRAI*, 19(2):165–182, 2005.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases".

In *Proceedings of the SIGMOD Conference, Montreal, Canada*, 1996.